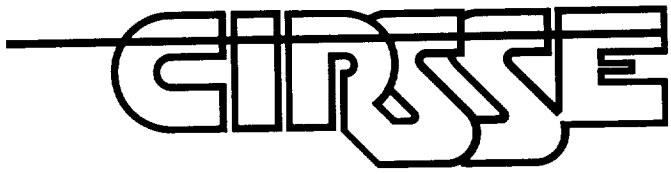

Center for Intelligent Robotic Systems for Space Exploration

Rensselaer Polytechnic Institute
Troy, New York 12180-3590

A COORDINATION MODEL FOR INTELLIGENT MACHINES

By:

Fei-Yue Wang
G.N. Saridis

Department of Electrical, Computer and Systems Engineering
Department of Mechanical Engineering, Aeronautical Engineering & Mechanics
Rensselaer Polytechnic Institute
Troy, New York 12180-3590

CIRSSE Document #15

A COORDINATION MODEL FOR INTELLIGENT MACHINES

FEI-YUE WANG AND GEORGE N. SARIDIS

JANUARY 1989

NASA Center for Intelligent Robotics Systems for Space Exploration

Robotics and Automation Laboratories
Department of Electrical, Computer, and Systems Engineering
Rensselaer Polytechnic Institute

Troy, New York 12180, U.S.A.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The purpose of this research is to develop an analytical model for the Coordination Level of Intelligent Machines, which, with the mathematical formulation for the Organization Level and the well developed control theory, would complete a mathematical theory for Intelligent Machines. The current progress toward such an analytical model is presented in this proposal. The framework of the Coordination Level investigated consists of a dispatcher, a set of coordinators. A formal model, called *coordination structure*, has been developed to describe analytically the information structure and information flow for the coordination activities in the Coordination Level. Specifically, the coordination structure offers a formalism to

- Describe the language (or task plans) translation characteristics of the dispatcher and the coordinators,
- Represent the individual process within the dispatcher and the coordinators, especially their concurrency and conflict,
- Specify the cooperation and connection among the dispatcher and the coordinators,
- Perform the process analysis such as deadlock-free, boundedness, etc, for the Coordination Level,
- Provide a control and communication mechanism for the simulation and real-time monition of the task processes in the Coordination Level.

A simple scheduling procedure for the task scheduling in the coordination structure is suggested. The task translation in the coordination structure is achieved by probabilistic learning processes. The learning processes are measured with Entropies and their convergence is guaranteed. A case study for a simple intelligent manipulator system is described, where a simple coordination structure with one dispatcher and three coordinators is built. The simulation of the task processes performed verifies the soundness of the theoretical results developed so far. Finally, we summarize the results with conclusions and give suggestions for the future research focuses on the modeling of Coordination Level of Intelligent Machines.

1. Introduction

In this early stage in the development of Intelligent Machines, methodological issues are both open and central. Different ideas for the formalization of the definitions and the structure of Intelligent Machines have been proposed by and debated among various researchers [Albus 1975, Saridis 1977, Bejczy, Meystel, Stephanou, Pao, 1986, Vamos 1987, Antsaklis et al 1988]. The approach proposed by Saridis (1977) can be thought as the result of the intersection of the three major disciplines of Artificial Intelligence, Operation Research, and Control Theory.

The structure of Intelligent Machines is defined by Saridis (1977, 79-80, 83, 85, 87) to be the structure of Hierarchically Intelligent Control Systems, composed of three levels hierarchically ordered according to the principle of ***Increasing Precision with Decreasing Intelligence (IPDI)*** [Saridis and Stephanou 1977, Saridis 1988], namely: the **Organization Level**, *performing general information processing tasks in association with a long-term memory*; the **Coordination Level**, *dealing with specific information processing tasks with a short-term memory*; and the **Execution Level** which *performs the execution of various tasks through hardware using feedback control methods* (Figure 1.1). A mathematical theory for Intelligent Machines has been presented in a recent paper by Saridis and Valavanis (1988), where the mathematical formulation for the Organization Level was developed.

The Coordination Level of an Intelligent Machine is an intermediate structure serving as an interface between its Organization Level and Execution Level for dispatching organizational information to execution devices. Its objective is the actual formulation of the

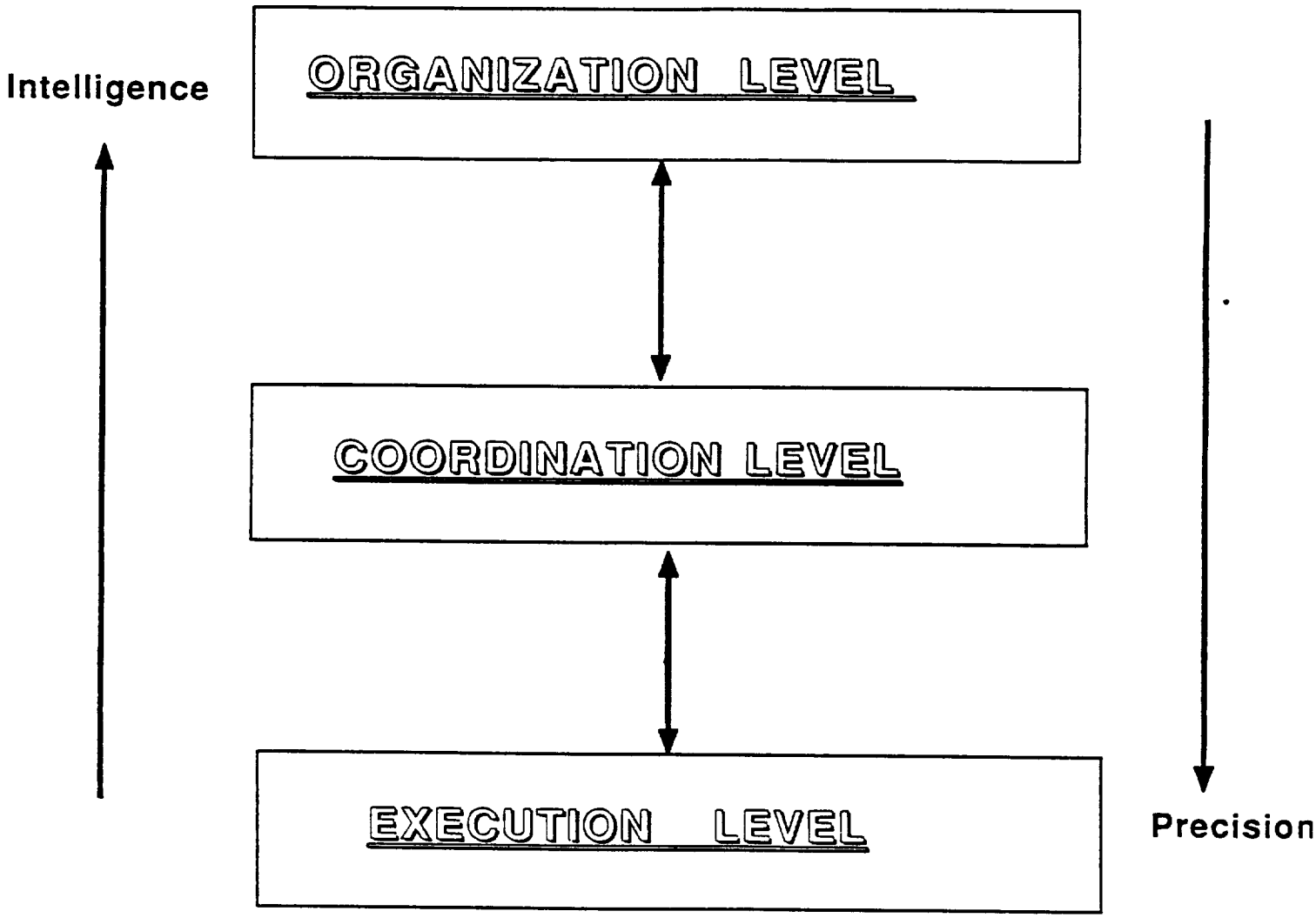

Fig.1.1. The Structure of Intelligent Machines

control problem associated with the most probable complete and compatible plan formulated by the Organization Level that will execute in real-time the requested job [Saridis and Valavanis 1988]. The purpose of this research is to develop an analytical model for the Coordination Level of Intelligent Machines, which, with the mathematical formulation for the Organization Level and the well developed control theory, would complete a mathematical theory for Intelligent Machines. It should be noted that a model for the Coordination Level is not required to represent how the coordination activities are actually realized in detail, however, any valid model should at least provide a control and communication mechanism for the coordination. This control and communication mechanism will enable the establishment of the information structure which specifies the necessary precedence relationship for the relevant information processing in the Coordination Level, and the formulation of the information flow which characterizes the actual decision-making activities for the achievement of the coordination objective.

This report presents the current progress toward such analytical model. The framework of the Coordination Level investigated consists of a dispatcher, a set of coordinators. A formal model, called *coordination structure*, has been developed to describe analytically the coordination activities among the dispatcher and the coordinators of a Coordination Level [Wang and Saridis 1988]. The use of this coordination structure enables us to

1. Describe the language (or task plans) translation characteristics of the dispatcher and the coordinators.

2. Represent the individual process within the dispatcher and the coordinators, especially their concurrency and conflict .
3. Specify the cooperation and connection among the dispatcher and the coordinators .
4. Perform the process analysis such as deadlock-free, boundedness, etc for the whole Coordination Level.
5. Provide a control and communication mechanism to be used for simulating and real-time monitoring the task process in Coordination Level.

Points 1 and 2 are accomplished by using *Petri net transducers* as the models for the dispatcher and the coordinators. A Petri net transducer is capable of performing the language translation and, like a Petri net, can describe the parallelism and conflictness (point 2). The cooperation and connection among dispatcher and coordinators are specified by the connection points and the receiving and sending mappings of the coordination structure (point 3). Point 4 is realized within the context of Petri net theory, since various concepts and analysis methods have developed for Petri nets to deal with the deadlock, boundedness, and other process properties. The standard execution rule in Petri net theory provides the base for the construction of Petri net controllers, which can be used to control and monitor the coordination processes in the dispatcher and the coordinators in real-time (point 5).

The report is organized into seven chapters. Chapter 2 reviews the distributed problem solving system theory. Three major approaches for the distributed problem solving systems in Distributed Artificial Intelligence and the several formal models for the

distributed systems in computer science and control theory are discussed in this chapter. Petri net and Petri net language theory is introduced in Chapter 3, where the basic concepts of Petri net, the closure properties of Petri net language and its relationship with other formal languages, are described through the definitions and examples. Chapter 4 presents the major results of this report, the definition and properties of the coordination structure. A uniform architecture for the dispatcher and the coordinators, the Petri net transducer models and their synchronous compositions are included in this chapter. Chapter 5 investigates the task scheduling and task translation in the coordination structure. A simple scheduling procedure based on the execution rule of Petri nets is suggested. The task translation is achieved using a probabilistic method with learning ability. The learning process is measured by Entropy and the learning convergence theorems are given. A case study for a simple intelligent manipulator system is described in Chapter 6, where a coordination structure is built for a dispatcher with three coordinators and the simulation of the task process in this coordination structure is performed. Finally, Chapter 7 summarizes the report with conclusions and gives suggestions for the future research focuses on the Coordination Level of Intelligent Machines.

2. Literature Review on Distributed Problem Solving Systems

The Coordination Level of Intelligent Machines is inherently a distributed problem-solving system, and, as all the distributed problem-solving systems, the key issue in the Coordination Level is the mechanism of the coherent control and communication of various processes in the system [Yang et al 1985, Decker 1987, Saridis 1988]. Considerable amount of work has been done during the past two decades for the distributed problem-solving. Basically, two different kinds of approaches have been used: the approach in Distributed Artificial Intelligence (DAI) and the approach in the theoretical computer science and control theory. The approach taken in DAI is more emphasized in the development of the actual distributed problem-solving systems, much of the effort is devoted into the system programming. One the other hand, the approach taken in the theoretical computer science and control theory is more interested in the specification of the formal process of distributed problem-solving systems, its emphasis is on the building of the formal models and the analysis of the properties of the systems. We will review the some key methods and models in these two approaches briefly in the following two sections .

2.1 Distributed Problem Solving Techniques in Distributed Artificial Intelligence

Several approaches for the coordination among the cooperating nodes of a network system have been suggested in the DAI. The three major important approaches are called multi-agent planning, negotiation, and the functionally-accurate, cooperative (FA/C) approaches. In the *multi-agent planning* approach, the agents typically choose an agent

from among themselves (perhaps through negotiation) to solve their planning problem and send this agent all pertinent information. The planning agent forms a multi-agent plan that specifies the actions each agent should take and the planning agent distributes the plan among the agents. Since the multi-agent plan is based on a global view of the problem, the important interactions between agents can be predicted and synchronized around [Corkill 1979, Futo and Gergely 1983, Georgeff 1984]. The main problem with the multi-agent planning approach is that achieving a global view of the problem might be time consuming and communication intensive, and the performance of the entire agent system depends on the planning agent and would be compromised if that agent fails. In the *negotiation* approach [Smith 1981a, Smith and Davis 1981, 1983], a node will decompose a problem task into some set of subtasks and will assign these subtasks to other nodes (for parallel execution) based on a bidding protocol [Smith 1981b]. Since nodes may have different capabilities, the bidding protocol allows a subtask to be assigned to the most appropriate available node However, since the nodes that are already working on subtasks are not available to bid until they have finished their tasks, it may cause the problem that a node which is awarded one subtask may thus be unavailable to perform a subsequently formed subtask despite being the best node for the task. If the node had been able to predict that a more coherent subtask might soon be formed, the node would not have bid on the earlier subtask so that it would be available later. The inability of nodes to make such predictions can therefore cause incoherence in the problem solving system: the nodes could make a more coherent team and improve their overall performance if they could assign subtasks to nodes better. In the *functionally-accurate, cooperative* (*FA/C*) approach to the distributed problem solving [Lesser 1981, Corkill 1983], nodes cooperate by generating and exchanging tentative, partial solutions based on their limited local views of the system

problem. By iteratively exchanging their potentially incomplete, inaccurate, and inconsistent partial solutions, the nodes eventually converge on an overall system solution. To cooperate coherently, the nodes would need to predict what partial solutions would be exchanged in the future and when, so that they could modify their problem solving activities to form compatible partial solutions. To make these predictions, each node needs to understand its own plans and the plans of the other nodes. Without this understanding, nodes may require much more time to converge on a solution since they may work at cross-purposes.

Prediction is therefore crucial for coherent cooperation. While multi-agent planning requires accurate predictions before it can form acceptable plans, the negotiation and the FA/C approaches can perform despite a lack of adequate predictions, but incoherence can degrade their performance.

One problem of applying the DAI approaches for the distributed problem solving in the mathematical theory of Intelligent Machines is the lack of the analytical models for these approaches. The emphasis of the DAI approaches is heavily on the system programming, and the system behavior is usually described declaratively in some loose terms, instead of being specified formally in terms of the well defined models. Therefore the formal models for the distributed coordination processes in the Coordination Level of Intelligent Machines have to be investigated and developed. However, the distributed problem solving techniques in DAI can provide useful guidances for the system organization and architecture of the Coordination Level.

2.2 Formal Models for Distributed Systems

The modeling of the distributed systems had been one of the central issues in computer science for a long time [Peterson 1973, Milner 1980, Hoare 1985]. In the control theory, some relevant works are conducted recently under names of Discrete Event Systems (DES) [Ramadge and Wonham 1982, Inan and Varaiya 1988], Discrete Event Dynamical Systems (DEDS) [Ho and Cassundras 1983], and Decision Schema [Saridis and Graham 1984].

Numerous theoretical models of concurrency, which can be used in DPS, have appeared in the computer science literature during the past 25 years: Dijkstra (1986a and b), Petri (1973), Campbell and Habermann (1974), Brinch-Hansen (1978), Hoare (1978), Hewitt (1979), Puneli (1979), Milner (1980), Steenstrup et al (1983), and Trakhtenbrot et al (1988), to mention a few. The purpose for which the various models have been developed includes the following: to facilitate the investigation of the behavior of concurrent processes, to aid in the designing of concurrent systems, to illustrate specific process synchronization problems, and to verify the correctness of parallel programs.

Milner (1980) has introduced an elegant calculus of synchronized communicating processes to express the behaviors of concurrent systems up to observation equivalence, and to provide various proof techniques. Note that Milner's approach to the semantics of concurrency is considerably more operational than the approach used in Milne-Milner (1979). The theory of communicating sequential processes developed by Hoare (1985) has been based on the ideas that input and output should be basic primitives of processing and

that parallel composition of communicating sequential processes is a fundamental process structuring method. It has been found that, when combined with a development of Dijkstra's guarded command, these ideas are surprisingly versatile. It should be noted that a major difference between the models of communicating sequential processes and the models of automata (e.g, the finite automata) is that in communicating sequential processes the primitive notion is that of a trace (behavior of the process) while the notion of state is a derived concept, whereas in the classical automata the primitive notions is that of state from which the trace is derived.

An elegant application of Hoare's theory in the modeling of the discrete event systems has been suggested by Inan and Varaiya (1988), where a new class of discrete event models called finitely recursive process (FRP) was introduced. It is claimed that the FRP offers a formalism for the discrete event systems that is superior to that of finite state machines. The basic idea in the FRP is to use a set of primitive functions to construct the general and complex process by using recursive equations. It is clear that the fixpoint theory has played the key role in the proof technique of FRP.

The FRP formalism seems to be very attractive and promising, however, since only preliminary work has been done in this direction, it is still not clear how far and how well FRP can go in incorporating the methodologies of the classical continuous control theory in the representation of discrete event systems.

One of the most popular tools in the modeling of concurrent process systems and discrete event systems, especially in manufacturing systems, is Petri net model [Al-Jaar and

Desrochers 1987, Chang et al 1987-88, Krauss 1988, Stotts 1988, Yau 1988, Zhou et al 1988]. The major advantage of Petri net model is that it can represent the commonsense logic formally in a quite natural way. The relatively straightforward graphical representation and the various available analytical tools for analyzing the system structure properties are also very helpful in using Petri net to model the systems. Much work has been done in applying Petri net models in the modeling of robotic system and process control [Bourbakis 1987, Cai 1987, Komoda 1984, Krogh 1988], decision-making [Ghalwash 1987-88, Levis 1988], performance evaluation [Narahari 1987, Al-Jaar and Desrochers 1988]. However, it should be noted that, upon to this point, no application of Petri net models in the specification and analysis of the distributed testbeds developed in robotic systems and process control, say, [Hayward 1988, Ionescu 1988, Karsai 1988, Lee 1987, Lee 1985,88, Pang 1988], has been made.

The focus of this research is on using Petri net to model and analyze the coordination processes in Coordination Level of Intelligent Machines. It is interesting to investigate the possibility and advantage of using the more general FPR formalism for this purpose, and this will be conducted in the next step.

3. Petri Nets and Petri Net Languages

In this chapter we introduce Petri net and Petri net Language theory. Section 3.1 is about the basic concepts of Petri nets, and Section 3.2 is on the Petri net language theory. These materials are quite standard and come mostly from Peterson (1981), except some modifications on the notations and the definition of Petri net language.

3.1 Basic Concepts of Petri Nets

In this section, we give a brief introduction to Petri net theory through some definitions and examples. Note that although only the ordinary Petri nets are treated here, we will use the concepts in the *colored Petri nets* freely later in our model for the Coordination Level, since it has been proved that as long as the number of colors is finite, the colored Petri net model is equivalent to a (much large) ordinary Petri net [Peterson 1980]. The detail descriptions of the colored Petri nets and another high level Petri net, the *predicate/transition nets*, can be found in [Jensen 1981, Genrich and Lautenbach 1981].

Petri nets are tools for modeling the dynamic behavior of discrete event systems. They consist mainly of two types of elements: places and transitions. The set of places represents the system' s states, and the transitions represent events which change the state of the system. A place can contain a non-negative integer number of tokens. The state of the system modeled by a Petri net is given by its marking, i.e., the number of tokens in each of its places. The system evolves by firing its transitions according the execution rule, as described below.

Definition 3.1: A *Petri net* (PN) is a quadruple N=(P, T, I, O) where:

1) P and T are finite sets of *places* and *transitions*, respectively, such that

$P \cap T = \phi$ and $P \cap T \neq \phi$,

2) I: $P \times T \rightarrow Z$ is the *input function*,

3) O: $P \times T \rightarrow Z$ is the *output function*,

where Z is the set of natural numbers.

A PN can be represented by a bipartite directed multigraph, the *Petri net graph*. Places are represented by *circles* and transitions by *bars*. There is an arc joining a place p to a transition t iff $I(p,t) \neq 0$, and p is called the *input place* of t. Analogously, there is an arc from a transition t to a place p iff $O(p,t) \neq 0$, and p is called the *output place* of t. Natural numbers I(p,t) and O(p,t) are called the weights of the arcs. Arcs are labeled with their weights. Labels will be omitted if the arc's weight is equal to one (Figure 3.1).

For convenience, let the places be numbered $p_1, p_2, \ldots, p_m$, m=|P|, in some unique way. We introduce two vectors $I(t) \in Z^m$ and $O(t) \in Z^m$ to represent the input and output function, that is, I(t) is a vector with its i-th coordinate being $I(p_i,t)$ and O(t) is a vector with its i-th coordinate being $O(p_i,t)$. Expressions $p \in I(t)$ and $p \in O(t)$ are used to indicate that p is an input place of t and p is an output place of t.

Definition 3.2: A *marking* m of a PN N is a function m: $P \rightarrow Z$. It gives the number of *tokens* contained in each place $p \in P$.

A token can be represented by a dot. Figure 3.1a shows a PN with its initial marking $m_0(p_1) = m_0(p_2) = 1$, $m_0(p_3) = 0$. The marking can be more briefly expressed as a column vector in Z^m, e.g., $m_0 = (1\ 1\ 0)^T$.

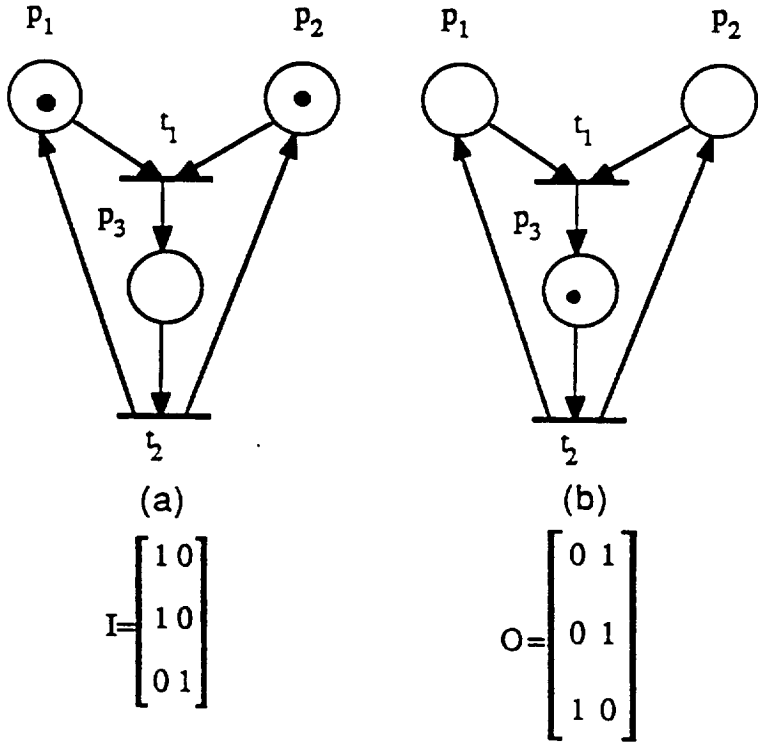

Fig. 3.1 A Marked PN (a) Before (b) After Firing t_2

Definition 3.3: A transition t is *enabled* wrt a marking m iff: $m \geq I(t)$, i.e., for all $p \in P$, $m(p) \geq I(p,t)$.

In figure 3a, only transition t_1 is enabled and, in figure 3b, t_2 is enabled.

Definition 3.4 (*execution rule*): *Firing* an enabled transition t consists of removing I(p,t) tokens from each input place p and adding O(p,t) tokens to each output place p. Let m_1 be the new marking resulted from firing t under the marking m_0, then, $m_1 = m_0 + O(t) - I(t)$, i.e., for all $p \in P$, $m_1(p) = m_0(p) + O(p,t) - I(p,t)$.

Figure 3.1b shows the marking of the PN N after firing the enabled transition t_1. The marking reached is $m_1 = (0\ 0\ 1)^T$.

For a PN N with marking m, two transitions t_1 and t_2 are said to be in *parallel* wrt m iff $I(t_1) + I(t_2) \leq m$. Two transitions are said to be in *conflict* wrt to m if $I(t_1) \leq m$, $I(t_2) \leq m$, and $I(t_1) + I(t_2) > m$. By the execution rule, two transitions in parallel can be fired simultaneously, however, only one of the two transitions in conflict can be fired and its firing will disenable the other. Figure 3.2 gives two examples of the parallel and conflict.

Let m be the marking reached from m_0 by applying the firing sequence s, $m_0 - s \rightarrow m$. If y is the count vector of s (i.e., y represents the number of times each transition has been fired in s), then m can be expressed by the *state equation*

$$m = m_0 + Ay$$

where $A = O - I = [a(p,t)]$, $a(p,t) = O(p,t) - I(p,t)$, is called the *incidence matrix* of the Petri net. In the example of figure 3.1,

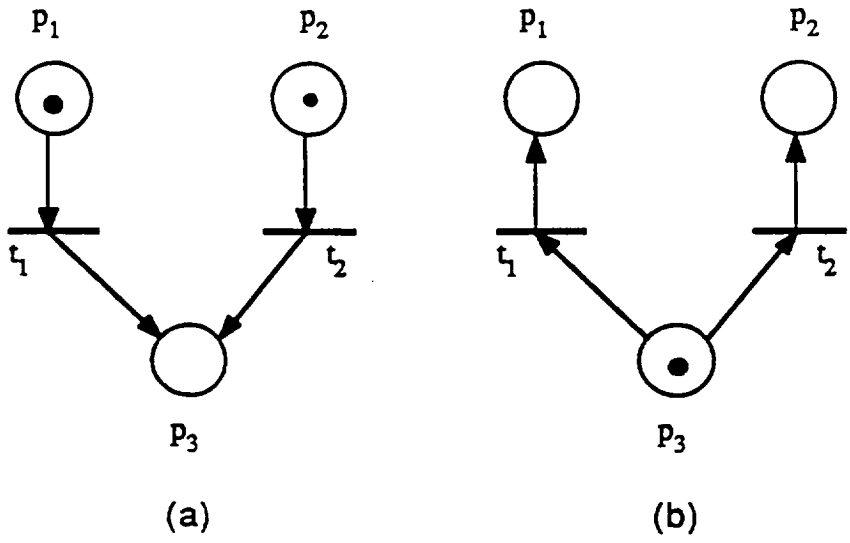

Fig. 3.2 Two Transitions are in (a) Parallel and (b) Conflict

$$A=\begin{bmatrix} -1 & 1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix}$$

and if $s=t_1t_2t_1$, i.e., then $y=(2\ 1)^T$, leads to the marking $m=m_0 + Ay =(0\ 0\ 1)^T$.

The *state* of a PN is defined by its marking. The firing of a transition represents a change in the state of the PN by a change in the marking of the net. The state space of a PN with m places is the set of all markings, i.e., Z^m. The change in state caused by firing a transition is described by the next-state function defined as

Definition 3.5: The *next-state function* δ: $Z^m \times T \rightarrow Z^m$ for a Petri net N=(P,T, I,O) with marking m and transition $t \in T$ is defined iff t is enables by m. If $\delta(m,t)$ is defined, then, $\delta(m,t)=m + O(t) - I(t)$.

The set of all markings reached from a marked PN is called the reachability set of the marked PN, which is formally defined as

Definition 3.6 (*reachability set*): The *reachability set* R(N, m) (or, R(m) when N is clear) for a PN N with initial marking m is the set of all markings of N which can be reached from m by firing a finite number of transitions of N.

Let T^* denote the set of strings over T and λ denote the *empty* string (the string with zero length). The definition of the next-state function δ can extend to strings in T^* in a

obvious way: $\delta(m,t\alpha)=\delta(\delta(m,t),\alpha)$, $\delta(m,\lambda)=m$, $t\in T$, $\alpha\in T^*$. We will use the next-state function under this extended definition.

Two sequences result from the execution of a PN: the *sequence of markings* (m_0, m_1, m_2, ...) and the *sequence of transitions* (t_{i_0}, t_{i_1}, t_{i_2}, ...). These two sequences are related by the relationship $\delta(m_k,t_{i_k})=m_{k+1}$ for k=0, 1, 2, ... Both of these sequences provide a record of the execution of the PN, one is about the state transformations, and the other is about the corresponding system actions. Corresponding to reachability set, we use L(N, m) (or, L(m) when N is clear) to denote the *set of all possible sequences of transitions* for a Petri net N with the marking m. Note that, for a system modeled by a PN N, L(N,m) characterizes the system behavior in the sense that *all possible sequences of the system actions are specified by the sequences in L(N,m)*, which is extremely importance in system designing, analyzing, and implementing.

The following are some of the properties and questions that have been studied in the literature about Petri nets [Al-Jaar and Desrochers 1987]. Later we will use these properties to analyze the behavior of the Petri net model for Coordination Level.

1) A *deadlock* in a PN occurs when a marking is reached where no transitions in the net can be fired from that point on.
2) A PN is *live* wrt a marking m if, for any marking in R(m), it is possible to fire any transition in the net. Liveness guarantees the absence of deadlocks.
3) A PN is *reversible* or *proper* wrt a marking m if for every $m'\in R(m)$, $m\in R(m')$. Reversibility guarantees that the system modeled by the PN can re-initialize itself. This is very important for automatic error recovery.

4) A PN is *bounded* wrt a marking m if there exists a finite number k such that for any marking in R(m) the number of tokens in each place of the PN under that marking is less than k. When k=1, the PN is *safe*.

It is easy to show that the PN in figure 3 is live, reversible, and safe, therefore it is deadlock-free and bounded.

3.2 Petri Net Languages and Their Closure Properties

The Petri net language (PNL) defined by a PN is intend to characterize the behavior of the system modeled by the PN through the specification of action sequences of the system. Although various formulations for PNL have been suggested [Peterson 1976], in order to be consistent with the definition of Petri net transducer introduced later, we present a general unified definition for PNL which will reduce to the existing formulations by placing the appropriate restrictions.

The PNL is defined as

Definition 3.7: A *Petri net language* (PNL) generated by a *labeled Petri net* $\gamma=(N, \Sigma, \beta, \mu, F)$ is a set of strings over Σ defined by

$$L(\gamma)=\{\beta(\alpha)\in \Sigma^* \mid \delta(\mu,\alpha)\in F\}$$

where

(i) $N=(P, T, I, O)$ is a *Petri net* with the *initial marking* μ;

(ii) Σ is a finite *alphabet;*

(iii) $\beta: T \rightarrow (\Sigma\cup\{\lambda\})$ is a *labeling function*;

(iv) $F \subset R(\mu)$ is a set of the *final markings*.

Different types of PNL can be obtained by considering various restrictions placed on the labeling function β and the final marking set F. The following three kinds of labeling function and four kinds of final marking set are suggested in the literature:

Labeling function:

(i) *free labeling function* β: $\beta(t)\neq\lambda$, $\beta(t_1)\neq\beta(t_2)$ if $t_1\neq t_2$;

(ii) *λ-free labeling function* β: $\beta(t)\neq\lambda$;

(iii) *λ-transition labeling function* β: no restriction on β;

Final marking set:

(i) F is *L-type* if F is a finite set of markings in $R(\mu)$;

(ii) F is *G-type* if $F=\{m\in R(\mu)|\ m\geq m_i$ for some i, i=1, ...,n$\}$;

(iii) F is *T-type* if $F=\{m\in R(\mu)|$ m is a deadlock marking of PN$\}$;

(iv) F is *P-type* if $F=R(\mu)$.

A labeled Petri net with a free labeling function is called a *free-labeled* Petri net, a labeled Petri net with a L-type final marking set is called a *L-type labeled* Petri net, and the corresponding PNL is called a *L-type* Petri net language, and so on. There exist 12 classes of PNL resulting from the cross product of the three types of labeling functions and the four types of the final marking specification. Figure 3.3 gives the known relations among the classes of PNLs.

In Figure 3.4, the initial marking of the labeled PN is $\mu=(1, 0, 0, 0)^T$, and each transition t is labeled by the free labeling function $\beta(t)$. Then

(i) If $F=\{(0, 0, 1, 0)\}$, the L-type PNL is $\{a^ncb^n|\ n\geq 0\}$;

(ii) If $F=\{m|\ m\geq(0, 0, 1, 0)\}$, the G-type PNL is $\{a^mcb^n|\ m\geq n\geq 0\}$;

(iii) The T-type PNL is $\{a^mcb^nd|\ m\geq n\geq 0\}$;

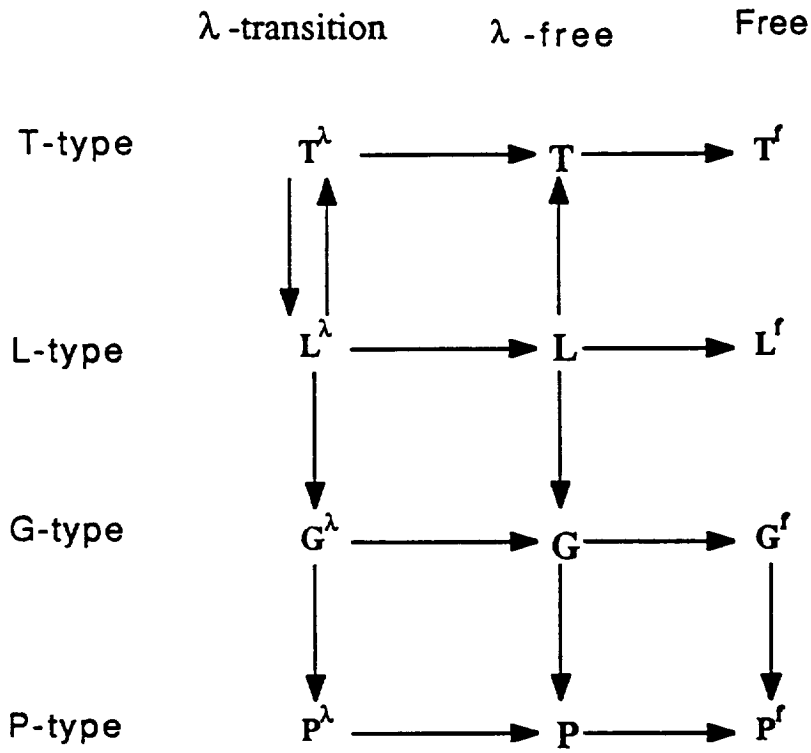

Fig. 3.3 The Known Relations Among the Classes of PNLs
(An arc from a class A to a class B means that class A contains class B)

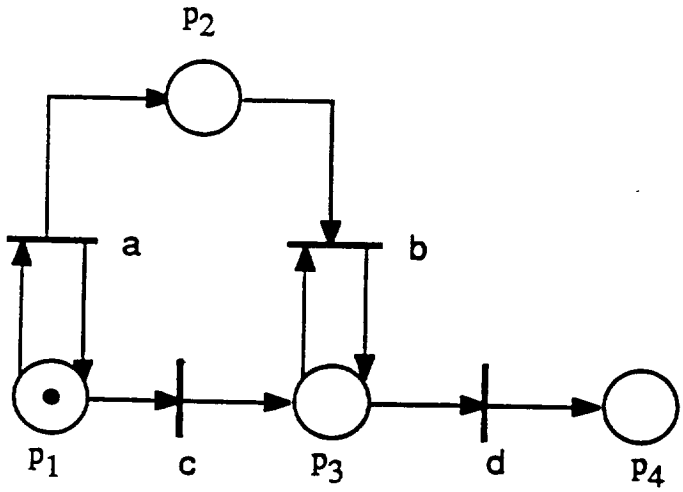

Fig. 3.4 A Free labeled Petri Net

(iv) The P-type PNL is $\{a^m | m \geq 0\} \cup \{a^m c b^n | m \geq n \geq 0\} \cup \{a^m c b^n d | m \geq n \geq 0\}$.

We will just consider L-type PNL in this report. It can be shown that *every L-type PNL can be generated by a standard form Petri net* defined as [Peterson 1981]:

Definition 3.8: A labeled PN $\gamma=(N, \Sigma, \beta, \mu, F)$ with PNL $L(\gamma)$ in *standard form* satisfies the following properties:

(i) The initial marking μ consists of exactly one token in a *start place* p_s and zero tokens elsewhere. $p_s \notin O(t)$ for all $t \in T$

(ii) There exists a *final place* p_f such that

(a) $F=\{p_f\}$ if $\lambda \notin L(\gamma)$ or $F=\{p_s, p_f\}$ if $\lambda \in L(\gamma)$,

(b) $p_f \notin I(t)$ for all $t \in T$,

(c) $\delta(m,t)$ is undefined for all $t \in T$, and $m \in R(\mu)$ which have a token in p_f (i.e., $m(p_f)>0$).

The use of the standard form labeled Petri net usually can simplify the analysis. The L-type PNL generated by the labeled PN in standard form of Figure 3.5 is $\{a^n b^n c^n | n \geq 0\}$. Since it can be showed easily by pumping lemma that $\{a^n b^n c^n | n>0\}$ is not a context-free language, the result indicates that PNL is not a subset of context-free languages. It has also be proved that the context-free language $\{xx^R | x \in \Sigma^*$ with $|\Sigma| \geq 2$, and x^R is the reversal of $x\}$ cannot be generated by any labeled PN, that is, PNs are not capable of remembering arbitrarily long sequences of arbitrary symbols. The two results together indicate that Petri nets are a new type of automata. However, it is easily to show that *a bounded Petri nets is equivalent to the finite state machines derived from its finite reachability set.* Neglecting the empty string λ, PNL is a strictly a subset of context-sensitive language [Peterson 1973]. The relation between PNL and other formal languages has been studied by several

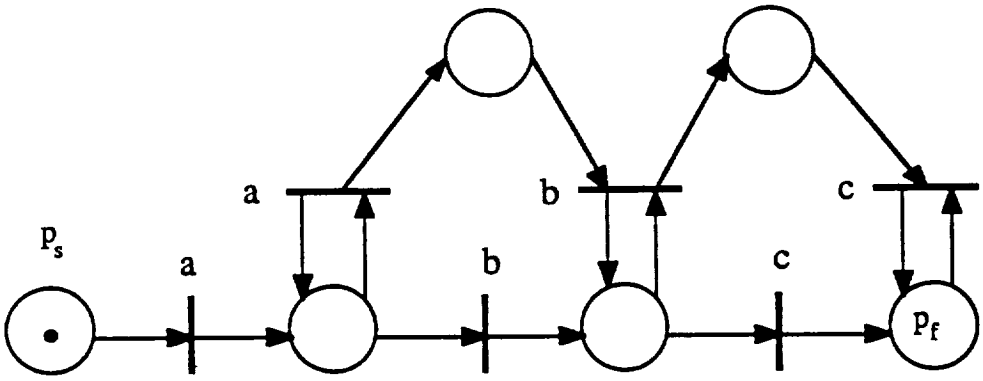

Fig. 3.5 A PNL $L(\gamma)=\{a^n b^n c^n \mid n>0\}$ Which Is Not A Context-Free Language

researchers [Reghizzi 1977, Vak 1981]. Figure 3.6 presents the relationship of PNL to the traditional classes of phrase structure languages.

The closure properties of PNLs can be summarized as follows:

(i) **Concatenation:** If L_1 and L_2 are PNLs, then
$L_1 L_2=\{x_1x_2 | x_1\in L_1, x_2\in L_2\}$ is a PNL.

(ii) **Union:** If L_1 and L_2 are PNLs, then
$L_1\cup L_2=\{x | x\in L_1 \text{ or } x\in L_2\}$ is a PNL.

(iii) **Intersection:** If L_1 and L_2 are PNLs, then
$L_1\cap L_2=\{x | x\in L_1 \text{ and } x\in L_2\}$ is a PNL.

(iv) **Reversal:** If L is a PNL, then
$L^R=\{x^R | x\in L\}$ is a PNL.

(v) **Concurrency:** If L_1 and L_2 are PNLs, then
$L_1||L_2=\{x_1||x_2 | x_1\in L_1, x_2\in L_2\}$ is a PNL.

(vi) **Substitution:** If L_1 is a regular language and L_2 is a PNL, then the result of substituting L_1 for a symbol in L_2 is a PNL.

The *reversal operator* and *concurrent operator* used in (iv) and (v) are defined as

$\lambda^R=\lambda$, $(ax)^R=x^Ra$, $a\in\Sigma$, $x\in\Sigma^*$;

$x||\lambda=\lambda||x=x$, $x\in\Sigma^*$, $(ax_1)||(bx_2)=a(x_1||bx_2) + b(ax_1||x_2)$, $a,b\in\Sigma$, $x_1, x_2\in\Sigma^*$;

It follows from (vi) that PNLs are closed under finite substitution and homomorphism. PNLs are not closed under *indefinite concatenation* (Kleene star) and the *general substitution*, and the closure under *complement* for the L-type PNLs is still an open problem.

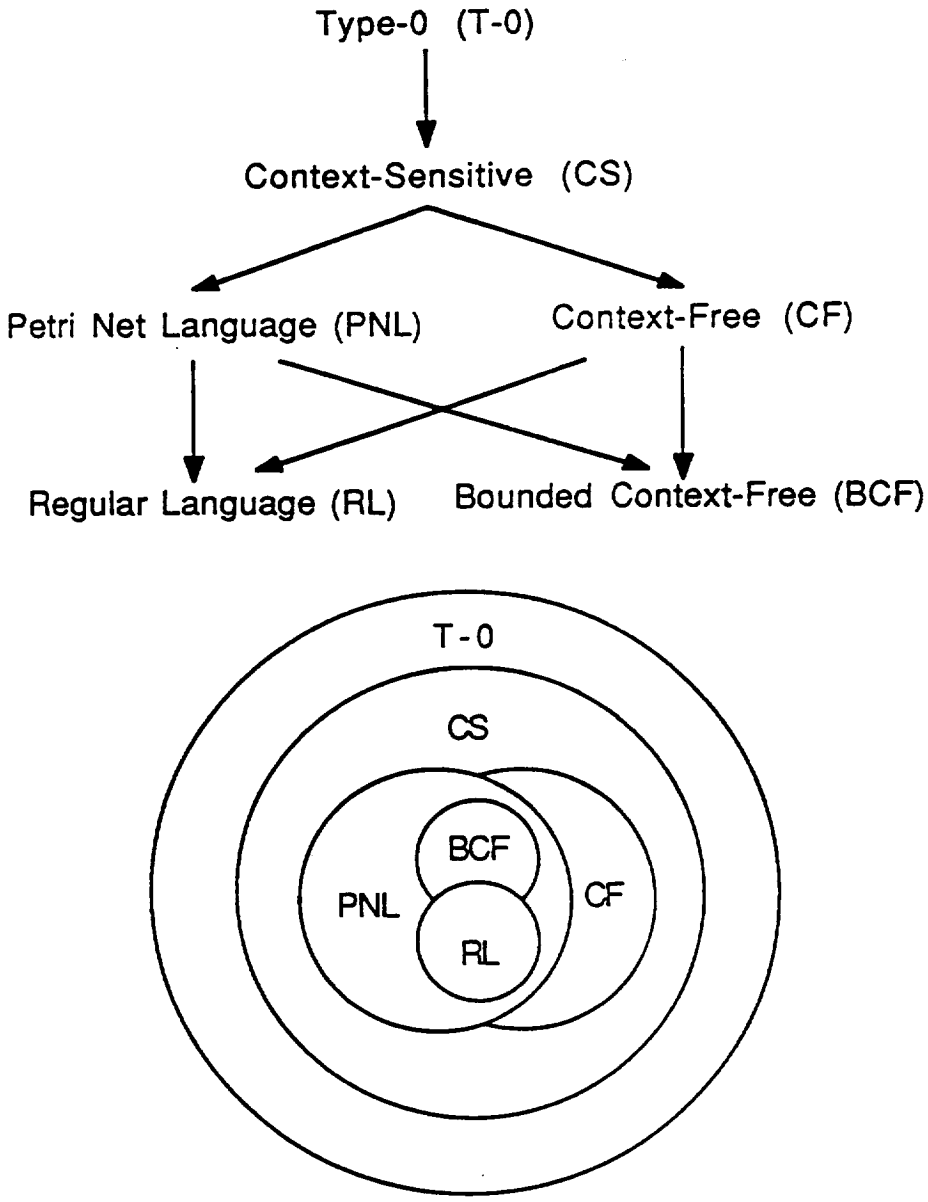

Fig. 3.6 PNL In The Hierarchy Of Formal Languages

4. The Model for Coordination Level: Coordination Structure

In this chapter, a formal model, called *coordination structure*, is introduced to describe mathematically the information structure and information flow in the Coordination Level of Intelligent Machines. Section 4.1 presents the framework for Coordination Level upon which the coordination structure is based. Section 4.2 introduces a new type of transducer, *Petri net transducer*, which will be used as the model for the dispatcher and the coordinators. Section 4.3 defined the *synchronous composition* of Petri net transducers. The coordination structure and its operation are described in the section 4.4. The analysis of the structural properties of the coordination structure is discussed in the section 4.5

4.1 The Framework of Coordination Level

The topology of the Coordination Level can be expressed by a tree structure CL consisting of a *dispatcher* D, the root, and a finite set of *coordinators* C, the subnodes,

$$CL=(D,C)$$

For convenience, let the coordinators be numbered C_1, C_2, ..., C_n, n=|C|, in some unique way (Fig. 4.1). It is assumed that for each coordinator there exists a bidirectional link connecting it to the dispatcher and there is no direct link between any two individual coordinators.

The dispatcher D centrally located above all coordinators will deal with the *control* and *communication* of the coordinators. It concerns primarily the questions of which coordinator(s) should be called when (*task sharing*) and which coordinator(s) should be informed by the current status of task execution (*result sharing*), given a sequence of *primitive events* (subtasks) by the organizer for some specific job. The control and communication can be achieved by translating the given sequences of primitive events into

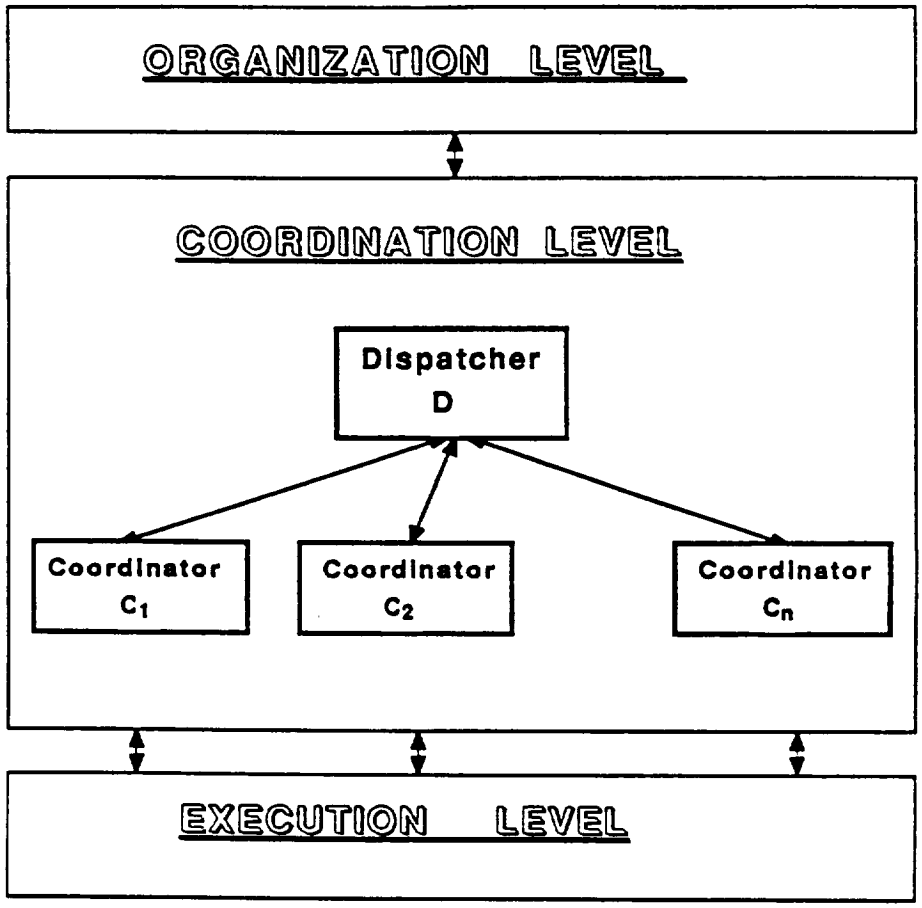

Fig.4.1 Topology of the Coordination Level

the sequences of *coordinator-oriented control actions* containing the necessary information, and dispatching them to the corresponding coordinators at the appropriate times, To this end, the dispatcher requires the following capabilities.

- A *communication facility* that allows the dispatcher to receive and send information from and to the Organization Level and coordinators.
- A *data processing ability* which describes the command information from the Organization Level and the feedback information from coordinators, updates the current system status, and provides information for the decision-making units of the dispatcher.
- A *task processing ability* which identifies the subtasks to be executed, selects the appropriate control processes for the corresponding coordinators, and formulates the feedback required by by the Organization Level.
- A *learning ability* that enables the dispatcher to improve its task processing ability and reduces uncertainties in decision-making and information processing as more task execution experience is obtained.

Each coordinator C_i centrally located above all devices associated with it will process the *operating* and *data passing* of the devices. The coordinator can be considered as an expert of deterministic functions in some specific field with the ability of selecting one among alternative actions that may accomplish the same subtask issued by the dispatcher in different ways according to the constraints imposed by the workspace model and timing requirements. The operating and data passing of the devices can be achieved by translating

the given coordinator-oriented control action sequences into *real-time hardware-oriented operation* sequences containing the necessary data, and sending them to the devices. Capabilities required by a coordinator are exactly the same as that for the dispatcher, but in a lower and more specific level. Figure 4.2 illustrates the translation process among the dispatcher and the coordinators. Note that the dispatcher and the coordinators actually have the different time scale, one step in the dispatcher may turn out to be many steps in the coordinators.

The coordinators have to cooperate under the supervision of the dispatcher in the sense that no one of them has sufficient ability and information to accomplish the entire task; mutual sharing of information is necessary to allow the dispatcher and the coordinators, as a whole, to attack the requested jobs.

The above description also indicates that the dispatcher and coordinators may have the identical organization at the different levels of specification. A uniform system architecture, consisting of a *data processor*, a *task processor*, and a *learning processor*, for the dispatcher and coordinators, is shown in Figure 4.3. This architecture is a direct extension of the decision module suggested by Graham and Saridis (1982).

The function of the data processor is to provide the information about the tasks to be executed and the current system status. It has been divided into three levels of description: *task description*, *state description*, and *data description*. The relation among the three levels is shown in Figure 4.4. In the task description, a list of subtasks to be executed from the upper level units is given. The state description presents the *preconditions* and the *postconditions* (i.e., the effects of execution) for the execution of each subtask and the system status in some abstract terms. In Petri net model, the preconditions and the postconditions can be described in terms of the input places and the output places,

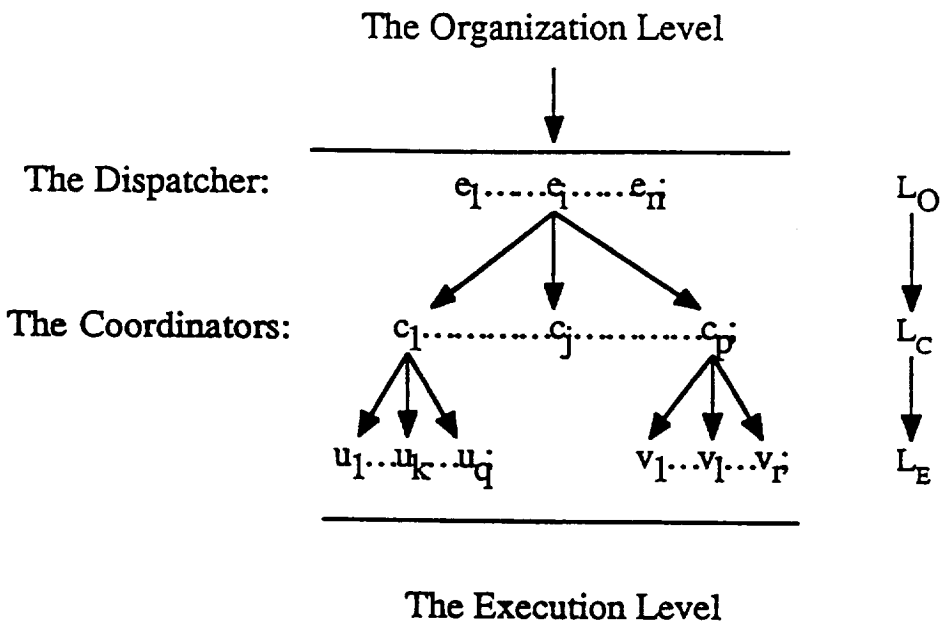

Fig. 4.2 The Language Translation Process $L_O \rightarrow L_C \rightarrow L_E$ in Coordination Level

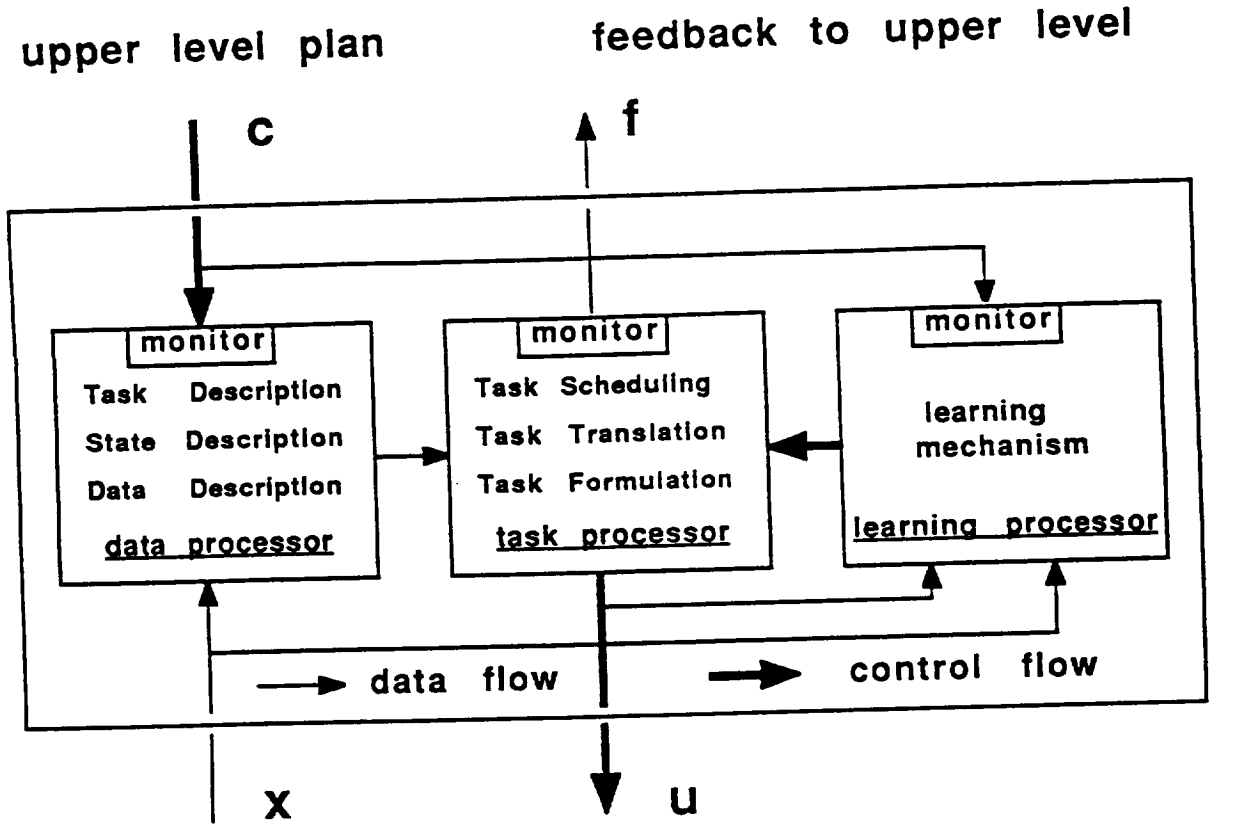

FIG. 4.3 A UNIFORM ARCHITECTURE FOR THE DISPATCHER AND COORDINATORS

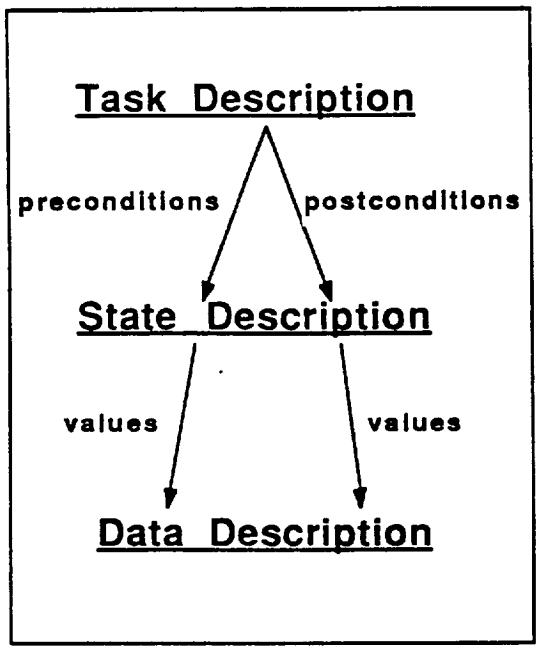

Fig. 4.4 Relation among the three Description Levels

respectively. The data description gives the actual value for the abstract terms used in the state description. Such an information organization is extremely useful for the hierarchical decision-making, as can be seen next in the task processor. The maintenance and update of the three level descriptions is manipulated by a *monitor* based on the information from the upper level and the feedback of task execution from the next level. The monitor is also responsible for the interconnection between the data processor and task processor.

The function of the task processor is to formulate the control command to the next level. The task processor employs a hierarchical decision-making consisting of three steps: *task scheduling*, *task translation*, and *task formulation*. The task scheduling identifies the subtask to be executed by checking the task description and the corresponding preconditions and postconditions contained in the state description without referring the actual values. If no subtasks can be executed, the task scheduling has to determine the internal operations which will make the preconditions for some subtasks to became true. The task translation decomposes the subtask or inter-operation into the control actions in an appropriate order based on the current system status. Finally, the task formulation responses to assign the actual data to the control actions by searching in the data description of the data processor, formulate the final complete control command, and send it to the next level. With the hierarchical information description, such a hierarchical decision-making should make the task processing fast and efficiently.

Upon the completion of all subtask required by the upper level, a *monitor* is called to organize the feedback information to the upper level in some specified form. The monitor is also responsible for the proper interconnection with the data processor and the learning processor.

The function of learning processor is to improve the performance of the task processor and to reduce the uncertainty in decision-making and information processing. Information used for the leering processor is indicated in Figure 4.3. Various learning mechanisms can be employed by the learning processor to achieve its function. A simple linear refinement learning algorithm is used for task translation in the section 5.2.

The fact that the dispatcher and coordinators have the identical system architecture but at different levels of specification (or abstraction) indicates that the Coordination Level has a *nested tree* topology [Meystel 1986]. This nested tree topology can be extended further to include the Execution Level.

The connection among the dispatcher and coordinators, i.e., the high level abstraction of communication among them, will be specified in term of a Petri net derived from the coordination structure, a formal model for the Coordination Level developed in the section 4.4.

4.2 Petri Net Transducers (PNTs)

As has been seen in the above discussion, the basic function of the Coordination Level can be viewed as the translation of the high level command language issued by the organizer to the low level operation language executed by the hardware. It seems that the automata, like finite transducer, pushdown transducer, and syntax-directed translation schemata, etc, used in the translation theory of formal language [Aho et al 1972] might offer us promising tools for the modeling of the Coordination Level. Unfortunately, this is not the case in genernally. Firstly, it is difficult to specify the connection among the dispatcher and coordinators, quite an important issue in the Coordination Level, by using automata. Secondly, it is inadequate to describe the concurrency of activities in the

Coordination Level by using automata, since the primitive notion *state* in automata is intend to represent the situation of the entire system modeled at some instance, whereas the notion of *place* in Petri net is just to describe the situation of a component of the system modeled. These two problems can be overcome by Petri net model, since it has been shown in both theory and applications that Petri net can enable, in a natural way, the specification of connection among the subsystems and the description of concurrency and conflictness in the systems processes. It is this observation leads to the motivation of developing a translation tool in term of Petri net, called Petri net transducer, as the basic model for the Coordination Level.

The Petri net transducer is defined as:

Definition 4.1: A *Petri net transducer* (PNT), M, is a 6-tuple,

$M=(N, \Sigma, \Delta, \sigma, \mu, F)$ where

(i) $N=(P, T, I, O)$ is a *Petri net* with the *initial marking* μ;

(ii) Σ is a finite *input alphabet;*

(iii) Δ is a finite *output alphabet;*

(iv) σ is a *translation mapping* from $T\times(\Sigma\cup\{\lambda\})$ to finite sets of Δ^*;

(v) $F\subset R(\mu)$ is a set of *final markings.*

A PNT can be pictured as shown in Figure 4.5. There are three parts to a PNT: an *input tape,* a *PN controller,* and an *output tape.* The behavior of a PNT can be conveniently described in terms of configurations of the PNT. A *configuration* of PNT M is defined as a triple (m, x, y) where $m\in R(\mu)$ is the current state (or marking) of the Petri net N; $x\in\Sigma^*$ is the input string remaining on the input tape with the leftmost symbol of x under the input head; $y\in\Delta^*$ is the output string emitted up to this point.

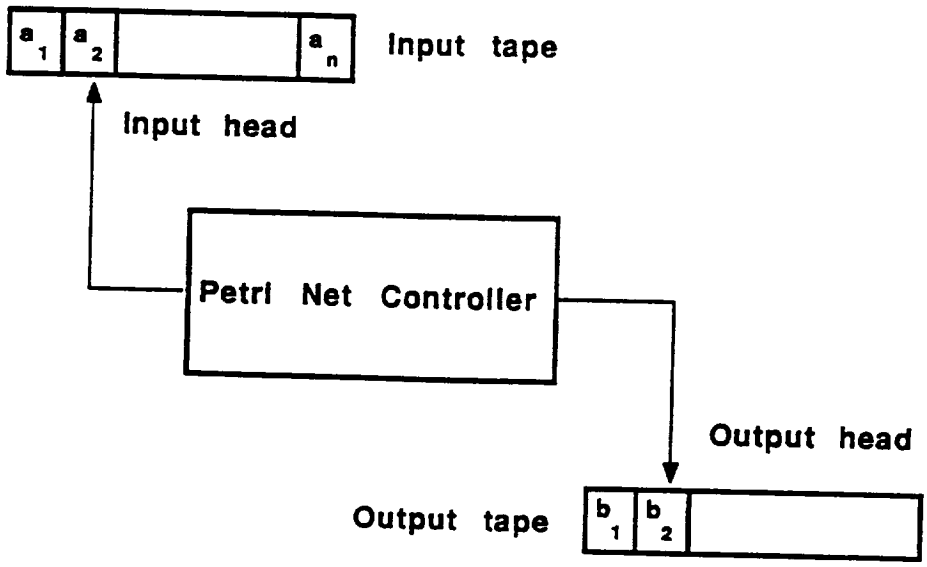

Fig. 4.5 Petri Net Transducer (PNT)

A *move* by PNT M is reflected by a binary relation $\Rightarrow_M$ (or $\Rightarrow$, when M is clear) on configurations. Specifically, for all $m \in R(\mu)$, $t \in T$, $a \in \Sigma \cup \{\lambda\}$, $x \in \Sigma^*$, and $y \in \Delta^*$ such that $\delta(m,t)$ is defined and $\sigma(t,a)$ contains $z \in \Delta^*$, we write

$$(m, ax, y) \Rightarrow (\delta(m,t), x, yz)$$

We will use $\Rightarrow^*$ to denote the *transitive* and *reflexive* closure of $\Rightarrow$. Note that the PNTs defined here allow λ-moves and are nondeterministic in both the firing of the next transition and the emitting of the next output string.

The *translation* defined by M, denoted $\tau(M)$, is the set $\tau(M)=\{(x,y) | (\mu,x,\lambda) \Rightarrow^* (m,\lambda,y)$ for some $m \in F\}$. y is said to be an *output* of x or x is the *input* of y iff $(x,y) \in \tau(M)$. The *input language* and the *output language* of M are defined as

$$\alpha(M)=\{ x | \text{there exists a } y \in \Delta^* \text{ such that } (x,y) \in \tau(M)\}$$

and

$$\omega(M)=\{ y | \text{there exists a } x \in \Sigma^* \text{ such that } (x,y) \in \tau(M)\}$$

respectively. Like other translation model, translation defined by a PNT will be called a *Petri net translation* or *Petri transducer mapping*.

A PNT *halts* at configuration (m, ax, y) when no transitions for which $\sigma(t,a)$ is defined are enabled at the current marking m. We call the m the *deadlock marking* of the PNT. When a deadlock marking occurs, the input string will be rejected. Note that a deadlock marking of PNT is not necessary a deadlock marking of its Petri net.

For any $x=x_1a_1a_2x_2 \in \Sigma^*$ with $(\mu,x,e) \Rightarrow^* (m,a_1a_2x_2,y)$, symbols a_1 and a_2 are said to be *parallel* (or in *conflict)* iff there exist t_1 and t_2 in T such that $\sigma(t_1,a_1)$ and $\sigma(t_2,a_2)$ are defined and t_1 and t_2 are parallel (or in conflict) with respect to the marking m. Two symbols in parallel can be translated simultaneously, however, only one of the two symbols in conflict can be translated and its translation will disenable the translation of the other.

Before to PNTs as the models for the dispatcher and the coordinators, we have to answer a fundamental question first: Can PNTs provide the consistent models for the dispatcher and the coordinators ? In other words, is there the possibility that the output languages of some PNTs cannot be further translated by any PNTs. If this is true, then it may happen that the tasks issued by the dispatcher cannot be processed by any kinds of coordinators, provided that PNTs are the only models for the dispatcher and the coordinators.

To answer this question we have to investigate the language properties of PNTs. For this end. let us consider a special class of PNTs, called *Simple PNT* (SPNT), with the property that for any $t \in T$ there exists one and only one $a \in \Sigma \cup \{\lambda\}$ such that $\sigma(t,a)$ is defined. The following theorem indicates the importance of this type of PNTs.

Theorem 4.1: *For any PNT M, there exists a SPNT M' such that $\tau(M')=\tau(M)$.*

Proof: Let N be the PN of $M=(N, \Sigma, \Delta, \sigma, \mu, F)$, $N=(P, T, I, O)$. For any $t \in T$, if there exist two different a_1 and $a_2 \in \Sigma \cup \{\lambda\}$ such that both $\sigma(t,a_1)$ and $\sigma(t,a_2)$ are defined, we then introduce a new transition t' with $I(t')=I(t)$, $O(t')=O(t)$ and modify σ in a way such that $\sigma(t',a_2)=\sigma(t,a_2)$, $\sigma(t',a)$ is undefined for all other $a \in \Sigma \cup \{\lambda\}$, and $\sigma(t,a_2)$ is undefined. Continuing this procedure until no transitions have more than two input alphabet in Σ for

which σ are defined, we will finally get a SPNT M'=(N', Σ, Δ, σ', μ, F) with N'=(P, T', I', O'). The construction of M' clearly shows that $\tau(M')=\tau(M)$. Q.E.D.

For a SPNT, we can define a *labeling function* β associated with σ as

$$\beta : T \rightarrow \Sigma \cup \{\lambda\},\ \beta(t)=a,\ \text{if } \sigma(t,a) \text{ is defined.}$$

As in PN language, we have three classes of labeling functions:*free* labeling function for which $\beta(t)\neq\lambda$, $\beta(t_1)\neq\beta(t_2)$ if $t_1\neq t_2$; *λ-free* labeling function for which $\beta(t)\neq\lambda$, $\beta(t_1)\neq\beta(t_2)$ if $t_1\neq t_2$; and *λ*-labeling function for which no constraints are imposed. For a SPNT M=(N, Σ, Δ, σ, μ, F), the labeled PN γ=(N, Σ, β, μ, F) is called the ***labeled Petri net underlying*** M.

We classify the PNTs according to the following four types of specifications of the final marking set F as in PN language.

(i) a PNT is *L-type* if F is a finite set of markings in R(μ);

(ii) a PNT is *G-type* if F={m$\in$R(μ)| m$\geq$m$_i$ for some i, i=1, ...,n};

(iii) a PNT is *T-type* if F={m$\in$R(μ)| m is a deadlock marking of PNT's PN};

(iv) a PNT is *P-type* if F=R(μ).

For SPNT there exist 12 classes of PN translations resulting from the cross product of the four types of the final marking specification and the three types of labeling functions. In this report we just consider L-type PNT.

Now we can characterize the language property of PNT by the following theorem:

Theorem 4.2: *The input and output language of a PNT are both PN languages.*

Proof: By theorem 4.1, we only need to consider the case for SPNTs. Let M=(N, Σ, Δ, σ, μ, F) be a SPNT and γ=(N, Σ, β, μ, F) be the labelled Petri net underlying M. The fact that input language is a PN language follows immediately from the equations $\tau(M)=L(\gamma)$.

Let γ'=(N, Σ, β', μ, F) be a labelled Petri net with a free labeling function β' and $L(\gamma')$ be its PN language. It is clear that $\omega(M)$ can be derived from $L(\gamma')$ by replacing the symbol $\beta'(t)$ in $L(\gamma')$ with any element of the finite set $\sigma(t, \beta'(t))$. Since PN language is closed under finite substitution, it follows that $\omega(M)$ is a PN language.
Q.E.D.

Theorem 4.2 guarantees that PNTs can be used as the consistent models for the dispatcher and the coordinators. Corresponding to the labelled PN in the standard form, we introduce

Definition 4.2: a SPNT M=(N, Σ, Δ, σ, μ, F) is said to be in the *standard form* iff the labelled PN γ=(N, β, μ, F) underlying M is in the standard form.

As in PN language, we can show that for any SPNT M, there exists a SPNT M' in the standard form such that $\tau(M')=\tau(M)$.

4.3 Synchronous Composition

To describe and specify the cooperation among the coordinators in the task processing, we introduce the synchronous composition operator in this section. First, we define inductively the *project* of a string x on a language L by

$$\lambda\uparrow_L=\lambda\ ,$$

$$(xa)\uparrow_L=\begin{cases}\text{undefined}, & \text{if } x\uparrow_L\notin L'\\ (x\uparrow_L)a, & \text{if } a\in\Sigma\\ x\uparrow_L, & \text{if } a\notin\Sigma\end{cases}$$

where $s\uparrow_L$ denote the project of s on language L, Σ is the alphabet of L, and L' is the closure of L, i.e., $L'=\{x|$ there is a $y\in\Sigma^*$ such that $xy\in L\}$. The synchronous composition is defined as

Definition 4.3: The *synchronous composition* of two PNTs

$$M_1=(N_1, \Sigma_1, \Delta_1, \sigma_1, \mu_1, F_1) \text{ and } M_2=(N_2, \Sigma_2, \Delta_2, \sigma_2, \mu_2, F_2)$$

with

$P_1\cap P_2=\phi$, $T_1\cap T_2=\phi$, and $\Sigma_1\cap\Sigma_2=\phi$ is a PNT M, denoted by

$M=M_1||M_2$, $M=(N, \Sigma, \Delta, \sigma, \mu, F)$ where

$$P=P_1\cup P_2,\ T=T_1\cup T_2,\ \Sigma=\Sigma_1\cup\Sigma_2,\ \Delta=\Delta_1\cup\Delta_2,$$

$$I(t)=\begin{cases}I_1(t), & t\in T_1\\ I_2(t), & t\in T_2\end{cases}\qquad O(t)=\begin{cases}O_1(t), & t\in T_1\\ O_2(t), & t\in T_2\end{cases}$$

$$\sigma(t, a)=\begin{cases}\sigma_1(t, a), & \text{if } (t,a)\in T_1\times\Sigma_1\\ \sigma_2(t, a), & \text{if } (t,a)\in T_2\times\Sigma_2\\ \text{undefined}, & \text{otherwise,}\end{cases}$$

$$\mu_1(p),\quad p\in P_1$$

$$\mu(p)= \begin{cases} \mu_2(p), & p\in P_2 \end{cases}$$

and

$$F=\{m\in R(\mu)|\ m=m_1\in F_1 \text{ for } p\in P_1 \text{ and } m=m_2\in F_2 \text{ for } p\in P_2\};$$

A move by $M=M_1||M_2$ can be represented as

$$(m_1, m_2, ax, y) \Rightarrow \begin{cases} (\delta(m_1,t_1), m_2, x, yz_1), & a\in \Sigma_1 \\ (m_1, \delta(m_2,t_2), x, yz_2), & a\in \Sigma_2 \end{cases}$$

where $z_1\in \sigma_1(t_1, a)$ and $z_2\in \sigma_2(t_2, a)$.

Synchronous composition for the case of $\Sigma_1\cap\Sigma_2\neq\phi$ involves a complex definition for the PN of the composition. But it is easy to describe the composition in this case in terms of configurations by defining

$$(m_1, m_2, ax, y) \Rightarrow \begin{cases} (\delta(m_1,t_1),\ \delta(m_2,t_2), x, yz_1z_2), & \text{or} \\ (\delta(m_1,t_1), \delta(m_2,t_2), x, yz_2z_1) \end{cases}$$

when $a\in \Sigma_1\cap\Sigma_2$, where $z_1\in \sigma_1(t_1, a)$ and $z_2\in \sigma_2(t_2, a)$.

By doing induction on the length of a string, we can prove the following important result:

$$\alpha(M_1||M_2)=\alpha(M_1)||\alpha(M_2)$$

where the operator " || " on the right side of the equation is the *concurrent operator* of two languages defined at the section 3.2.

Note that $\alpha(M_1||M_2)=\alpha(M_1)||\alpha(M_2)$ is not true for the case $\Sigma_1\cap\Sigma_2\neq\phi$. However, we have

$$\alpha(M_1||M_2)=\{x|x\uparrow_{\alpha(M1)}\in\alpha(M_1) \text{ and } x\uparrow_{\alpha(M2)}\in\alpha(M_2)\}$$

when $\Sigma_1\cap\Sigma_2\neq\phi$.

The synchronous composition can be extended to more than two PNTs by defining

$$M_1||M_2||\bullet\bullet\bullet||M_{k-1}||M_k=(M_1||M_2||\bullet\bullet\bullet||M_{k-1})||M_k$$

4.4 The Coordination Structures (CSs)

Now the model for the Coordination Level of Intelligent Machines is defined as:

Definition 4.4: A *coordination structure*, CS, is defined to be a 7-tuple,

$$CS=(D, C, F, R_D, S_D, R_C, S_C) \text{ where}$$

(i) $D=(N_d, \Sigma_o, \Delta_o, \sigma_d, \mu_d, F_d)$ is a PNT, called the dispatcher, with

$$N_d=(P_d, T_d, I_d, O_d) \text{ and } \Delta_o=\Delta_I\cup\Sigma_C,\ \Sigma_C=\cup_{i=1}^{n}\Sigma_c^i;$$

(ii) $C=\{C_1,C_2,...,C_n\}$ is the set of coordinators, $n\geq 1$. Each coordinator is a SPNT

$$C_i=(N_c^i, \Sigma_c^i, \Delta_c^i, \sigma_c^i, \mu_c^i, F_c^i) \text{ with } N_c^i=(P_c^i, T_c^i, I_c^i, O_c^i)$$

in the standard form except there is a transition $t_f^i\in T_c^i$, called the *final transition* of C_i with $\sigma_c^i(t_f^i,\lambda)=\lambda$ such that $I_c^i(t_f^i)=p_f^i$ and $O_c^i(t_f^i)=p_s^i$, p_s^i and p_f^i are the *start* and the *final* places of C_i , respectively. It is also assumed that

$$P_c^i \cap P_c^j = \phi,\ T_c^i \cap T_c^j = \phi,\ \Sigma_c^i \cap \Sigma_c^j = \phi,\ \Delta_c^i \cap \Delta_c^j = \phi, \qquad i \neq j,\ i,j = 1, \ldots, n;$$

(iii) $F = \cup_{i=1}^{n} \{f_I^i, f_{SI}^i, f_O^i, f_{SO}^i\}$ is the set of *connection points*: f_I^i is called the *input point* of C_i, f_{SI}^i the *input semaphore* of C_i, f_O^i the *output point* of C_i, and f_{SO}^i the *output semaphore* of C_i;

(iv) R_D and S_D are mapping from T_d to finite subsets of F, called the *dispatcher receiving* and *sending mapping*, respectively. R_D and S_D satisfy the following connection constraints:

(a) $(t, f_I^i) \in S_D \Leftrightarrow (t, f_{SI}^i) \in R_D$: which means that in order to send information to coordinator C_i, the dispatcher should check the input semaphore f_{SI}^i first;

(b) $(t, f_O^i) \in R_D \Leftrightarrow (t, f_{SO}^i) \in S_D$: which means that after receiving information from C_i, the dispatcher should reset the output semaphore f_{SO}^i.

(c) $(t, f_I^i) \notin R_D$ and $(t, f_{SO}^i) \notin R_D$: which means that the dispatcher cannot receive information from coordinators through f_I^i and f_{SO}^i;

(d) $(t, f_O^i) \notin S_D$ and $(t, f_{SI}^i) \notin S_D$: which means that the dispatcher cannot send information from coordinators through f_O^i and f_{SI}^i;

(e) if $(t, f_O^i) \in R_D$ then t is not initially enabled and that in any firing sequence which enables t, the number of the transitions t' with $(t', f_I^i) \in S_D$ is greater than the number of the transitions t' with $(t', f_O^i) \in R_D$: which means that before for a transition to receive the

execution result from a coordinator, there have to be other transitions activate the coordinator by sufficient times.
(f) for any f_I^i and f_O^i, there exists t, $t' \in T_d$ such that $(t, f_I^i) \in S_D$ and $(t', f_O^i) \in R_D$: which means every coordinator is connected with the dispatcher bidirectionally;

(v) R_C and S_C are mapping from $T_C = \cup_{i=1}^{n} T_C^i$ to finite subsets of F, called the *coordinator receiving* and *sending mapping*, respectively. R_C and S_C satisfy the following connection constraints:

(a) $(t, f_O^i) \in S_C \Leftrightarrow (t, f_{SO}^i) \in S_D$: which means that in order to send information through the output point, transition t has to check the output semaphore f_{SO}^i first;
(b) $(t, f_O^i) \notin R_C$ and $(t, f_{SI}^i) \notin R_C$: which means that the coordinators cannot receive information through f_O^i and f_{SI}^i;
(c) $(t, f_I^i) \notin S_C$ and $(t, f_{SO}^i) \notin S_C$: which means that the coordinators cannot send information through f_I^i and f_{SO}^i.

Figure 4.6 gives the configuration of the coordination structures. The input alphabet Σ_O, Σ_c^i, and the output Δ_c^i, i=1, ..., n represent the set of the *primitive events*, *primitive control actions*, and *primitive operations* defined at the Organization Level, Coordination Level, and Execution Level respectively. Δ_I is the set of the internal operations in the dispatcher. Each coordinator is associated with four connection points, called input point, input semaphore, output point, and output semaphore, respectively. The notation of connection point is similar to the concept of *port* in network theory, which has been used

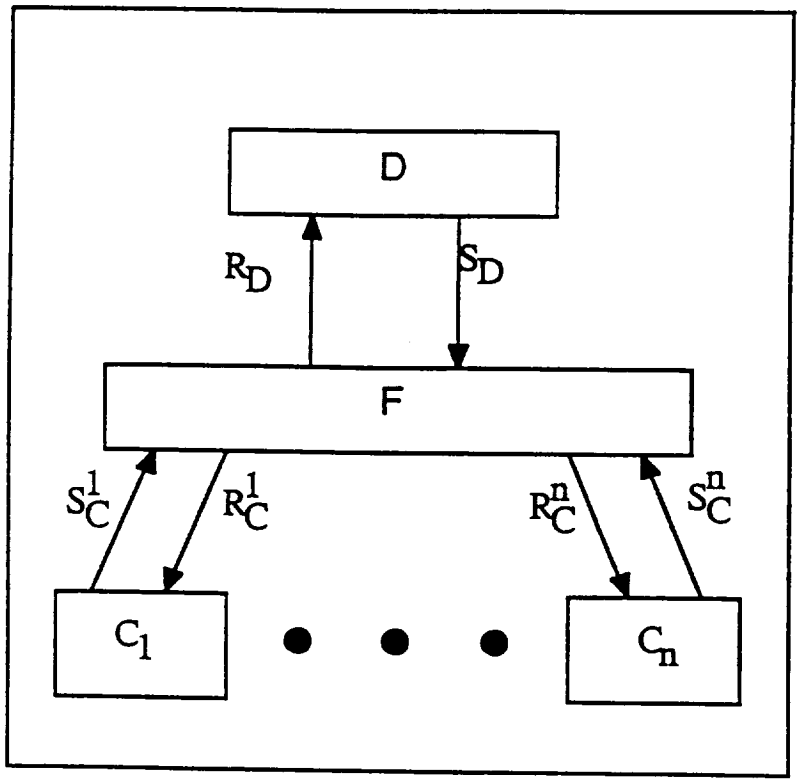

Fig. 4.6 The Coordination Structure

in the distributed sensor systems [I.Lee and Goldwasser 1985, Durrant-Whyte 1987] and the manufacturing process [Heukeroth and Nour Eldin 1988]. Specifically, a token in the input point of a coordinator indicates that a task has been issued to the coordinator, and the token also contains the necessary instruction (ordered coordinator–oriented control actions) and information for the task execution. The dispatcher can send a task execution command to a coordinator only if there is a token in the input semaphore of the coordinator, that is, a token in the input semaphore indicates the coordinator is available for the task execution. A token in the output point of a coordinator indicates that a task has been completed by that coordinator, and feedback information of task execution is contained in that token. A coordinator can send feedback of task execution to its output point only if there is a token in the output semaphore of the coordinator, which implies that the communication facility is ready for information transferring between the dispatcher and the coordinator. Once the corresponding transition in the dispatcher takes the feedback information from the output point, it will reset the output semaphore, by the connection constraints (or *connection protocol*) imposed.

The behaviors of the dispatcher and the coordinators are specified by the transition sequence sets $L(N_d, \mu_d)$ and $L(N_C^j, \mu_C^j)$, i=1, ..., n, respectively. As in program verification, where the behavior (all possible routine sequences) of a program is used to prove the correctness of the program and guide the implementation, the transition sequence sets $L(N_d, \mu_d)$ and $L(N_C^i, \mu_C^j)$ can be used for designing, analyzing, and implementing or simulating the models for the dispatcher and the coordinators. Let us define

$$T_d^i = \{t \mid t \in T_d \text{ and } (t, f_I^i) \in S_D\},\ i=1, \ldots, n$$

then, in order for the coordinator C_i to process the tasks from the dispatcher, the following relationship has to be satisfied

$$\alpha(C_i) \supseteq \cup\{\sigma_d(t, a)\uparrow_{(T_d^i)^*} \mid t \in T_d^i \text{ and } a \in \Sigma_O\},\ i=1, \ldots, n$$

that is, the coordinator C_i should be capable of processing all the possible task strings issued from the dispatcher. This relationship is guaranteed being able to be satisfied by the closure property of PNL under the union operation and the theorem 4.2.

The connection among the dispatcher D and coordinators C_i is the abstract specification of the actual communication facilities among the dispatcher and coordinators. It is clear that coordinators are not allowed to communicate with each other directly, since $R_C(T_C^i) \cap S_C(T_C^j) = \phi$, $i \neq j$ by the connection constraints imposed in the definition of the coordination structure. It should be noted that various complex connection patterns can be defined by using different receiving and sending mappings. One of the most basic connection patterns can specified as: (i) a coordinator can only access to its own input point and output point, and input semaphore and output semaphore; (ii) only one of the initially enabled transitions of a coordinator can receive information from its input points; (iii) only the final transition of a coordinator can send information to its output point. A coordinator structure with this type connection pattern is called a *simple coordination structure*. We will concentrate on the simple coordination structure in this report.

To describe the operation of the coordination structure CS, we first define the *Petri net underlying the CS* as N=(P,T,I,O), where

$$P = P_d \cup F \cup P_C,\ T = T_d \cup T_C,\ P_C = \cup_{i=1}^{n} P_C^i,\ T_C = \cup_{i=1}^{n} T_C^i;$$

$$I(t) = \begin{cases} I_d(t) \cup \{fl(t,f) \in R_D\},\ t \in T_d, \\ I_c^i(t) \cup \{fl(t,f) \in R_C\},\ t \in T_c^i; \end{cases}$$

$$O(t)=\begin{cases} O_d(t) \cup \{fl(t,f)\in S_D\}, & t\in T_d, \\ O_c^i(t) \cup \{fl(t,f)\in S_C\}, & t\in T_c^i. \end{cases}$$

The initial marking of N is defined to be

$$\mu(p)=\begin{cases} \mu_d(p) \text{ or } \mu_c^i(p), & \text{if } p\in P_d \text{ or } p\in P_c^i, \\ 1 & \text{if } p=f_{SI}^i \text{ or } f_{SO}^i, \\ 0 & \text{otherwise} \end{cases}$$

To start operation, the CS first receive a string in the task command language L_o, i.e., a sequence of primitive events, from the Organization Level and puts it on the input tape of the dispatcher D. The dispatcher D then begins the process of translating (or dispatching). Once a transition t of D with f_I^{i1}, ..., f_I^{is} as its output places in F is fired with respect to the current marking of the underlying PN N to execute the primitive event a, it will send the selected control action string $z\in \sigma_d(t,a)$ to each of input tapes of the coordinators C_{i_1}, ..., C_{i_s}, and cause them to operate synchronously, that is, activate the synchronous composition $C_{i_1} \| \cdots \| C_{i_s}$ to operate. Upon the completion of a subtask by a coordinator C_{i_k}, if the final transition t_f^{ik} is enabled with respect to the current marking of N at the time, it will fire by removing the token from p_f^{ik} and f_{SO}^{ik} and displacing a token to p_s^{ik} and f_O^{ik}. C_{i_k} becomes idle again and the execution result (feedback) is sent to the output point, which will be taken by the dispatcher to continue the process. Otherwise, if t_f^{ik} is not enabled, C_{i_k} has to wait until the output point f_O^{ik} (i.e., the communication facility for C_{i_k}) is available, since by the definition of the standard form SPNT, no other transitions in C_{i_k} can be enabled before t_f^{ik} removes the token from p_f^{ik}. Once the

configuration of D reaches (m_d, λ, y), $m_d \in F_d$, and the configurations of the coordinators are either $(\mu_c^i, \lambda, y_c^i)$ (which means the coordinator does not perform task for a while) or (m_c^i, λ, y_c^i), $m_c^i \in F_c^i$, the entire task process is completed successfully, and the requested job is accomplished.

A string $s \in \Delta_O^*$ is said to be executable by the CS if $(\mu_d, s, \lambda) \Rightarrow^* (m, \lambda, y)$, $m \in F_d$ and the final configuration of each of coordinators is either $(\mu_c^i, \lambda, y_c^i)$ or (m_c^i, λ, y_c^i), $m_c^i \in F_c^i$. It should be pointed out that not every string in $\alpha(D)$ is executable by the CS, because the additional connection restrictions imposed by the receiving and sending mappings. However, we can prove that a transition enabled in N_d after finite step firing can also be enabled in N after the same steps of firing (generally by a different path, however, see the proof of theorem 4.4). In any case, it has to be guaranteed that every string in L_o (a subset of $\alpha(D)$)should be executable by CS during its design phase.

The λ-move (the firings of transitions caused by $\sigma(t,\lambda)$) have the special physical significance. They may represent the internal operations occurred in the dispatcher or coordinators which are activated to provide the necessary information or resource for the continuity of the coordination process.

Clearly, the underlying PN N specifies the precedence relation among the activities in the dispatcher and coordinators and therefore defines the *information structure* of the CS. From the point of view of abstract execution of PN N, the string issued by the Organization Level can be considered as a path specification in the PN N_d, and, in turn, the string selected by the dispatcher D can be thought as the path specification in the PNs of the corresponding coordinators. This fact reveals once more the nested structure aspect of the Coordination Level modeled.

4.5 Some Structure Properties of the Coordination Structure

One of the merits offered by the above Coordination Level model is that the underlying PN N enables us to use the PN concepts and analysis methods to study the structure properties of the Coordination Level, such as liveness, boundedness, reversibility, consistency, repetitiveness, etc. The following two theorems present the results about the boundedness and liveness of the Coordination Level.

Theorem 4.3: The PN N underlying CS is bounded (safe) if all the PNs N_d, N_c^i, i=1, .., n are bounded (safe).

The proof is very simple. By the definition of receiving and sending mappings, for any $m \in R(N,\mu)$, $m(p) \leq 1$ if $p \in F$. Clearly, the PNs N_d, N_c^i, i=1, .., n are closed subnets of PN N, it follows immediately that the restriction of $R(N,\mu)$ on P_d is a subset of $R(N_d, \mu_d)$ and the restriction of $R(N,\mu)$ on P_c^i is a subset of $R(N_c^i, \mu_c^i)$. Therefore, the boundedness of N_d, N_c^i, i=1, .., n guarantees the boundedness of the underlying PN N. The claim is also true for safeness.

The boundedness of the underlying PN guarantees the structure stability of the CS. It can be shown, however, that for a PNT M with a bounded PN, we can define an equivalent finite transducer M' using the reachability set of the bounded PN such that $\tau(M')=\tau(M)$. This implies that the *input and output languages of the bounded PNT are actually the regular languages*, which indicates that the language complexity of PNTs with bounded PNs is very simple. Therefore, for some cases, the unbounded PNTs are required in the coordination structures.

Theorem 4.4: The PN N underlying CS is live if all the PNs N_d, N_c^i, i=1, .., n are live.

Proof: Considering that for every coordinator C_i there exists at least one transition in N_d which takes f_I^i as its output place in F, and that only the initially enabled transitions in N_c^i take f_I^i as their input place by the definition of the simple connection pattern, we only need to show that N_d as a subnet of the underlying PN N is live in order to prove N is live.

Let $m \in R(N,\mu)$ be an arbitrary marking, R(N,m,k) be the set of markings reached from m by firing at most k transitions in T_d, m_d and $R(N,m_d,k)$ be the restrictions of m and R(N,m,k) on P_d, and $R(N_d,m_d,k)$ be the set of markings reached from m_d by firing at most k transitions when N_d is considered as an independent PN. Let T(k) and T'(k) be the sets of transitions in T_d which are enabled under R(N,m,k) and $R(N_d,m_d,k)$ respectively. We claim that

$$R(N,m_d,k)=R(N_d,m_d,k),\ T(k)=T'(k)$$

When k=0, $R(N,m_d,k)=m_d=R(N_d,m_d,k)$, and, obviously, $T'(k) \supseteq T(k)$. Let $t \in T'(0)$ be an enabled transition. If $(t, f_O^i) \notin R_D$ for all i, then it is clear that t is also enabled by m, so $t \in T(0)$. If $(t, f_O^i) \in R_D$ for some i, then by the connection constraint (e) of (iv) in the definition 4.4, that in any firing sequence which enables t, the number of the transitions which activate C_i is greater than the number of the transitions which take the execution result from C_i, there should be tokens in f_O^i in the marking m, so t may fire under m. Therefore, $t \in T(0)$, hence $T(0) \supseteq T'(0)$, $T(0)=T'(0)$.

Assuming that $R(N,m_d,k)=R(N_d,m_d,k)$, $T'(k)=T(k)$, for $k \leq q$. $R(N,m_d,q+1) = R(N_d,m_d,q+1)$ follows immediately from $T'(q)=T(q)$. Since $T'(q+1) \supseteq T(q+1)$, by the same

argument used in the proof of T(0)=T'(0), we can show that $T(q+1) \supseteq T'(q+1)$, therefore $T(q+1)=T'(q+1)$.

Since N_d is live, every transition is possible to fire from m_d in N_d. From $R(N,m_d,k) = R(N_d,m_d,k)$, $T'(k)=T(k)$ for any k, we see that the same transition is also possible to fire from m through the same number of firings of transitions. Therefore, N is live. Q.E.D.

Even a transition in T_d can be enabled through the same number of firings of transitions of T_d in both PN N ans N_d from the same making. However, the firing sequences in N and N_d may be different. Especially, two transitions in T_d are parallel in N_d may no longer be parallel in N, since they may require inputs from the same coordinators.

For the construction of coordination structures, the methods of building the bounded and live Petri net models for manufacturing systems will be very useful. The step-wise refinement approach developed by Valette (1979), Suzuki and Murata (1980, 82-83), Zhou, DiCesare, and Desrochers (1988), as well as the hierarchical reduction analysis methods by Hyung et al (1985, 87) and Soog et al (1988) can be easily adapted for constructing the bounded and live Petri net models for the dispatcher and the coordinators.

5. Decision Making in Coordination Structure

We now investigate the problem of task scheduling and translation in the coordination structures. Section 5.1 describes a simple scheduling procedure based on the execution rule of Petri nets, and Section 5.2 presents a probabilistic method with learning for the task translation.

5.1 Task Scheduling

Task scheduling in the Coordination Level is the process of identifying the appropriate subtasks to be executed to complete the task issued by Organization Level. In CS model of Coordination Level, since the dispatcher and coordinators process their tasks simultaneously, the problem of task scheduling has to be dealed with distributedly. The task control for a special class of processes using the standard execution rule of Petri net has been studied by Komoda et al (1984), Murata et al (1986), and Crockett and Desrochers (1987). In the sequel we present a uniform scheduling procedure for the dispatcher and coordinators based on the execution rule of Petri net.

Let $M=(N, \Sigma, \Delta, \sigma, \mu, F)$ be a PNT representing the dispatcher or a coordinator. For any $a \in \Sigma$, we define

$$T(a)=\{t \mid \sigma(t, a) \text{ is defined}\},\ T_\lambda=T(\lambda)=\{t \mid \sigma(t, \lambda) \text{ is defined}\}.$$

Two queues, Q_T and Q_D, are used to in the scheduling procedure to record the task processing. Q_T stores the unexecuted subtasks, and Q_D the subtasks which are delayed to be executed due to that the transitions processing them are not be enabled at the appropriate time. Let First(Q) be a function which returns the first element of Q and deletes this element from Q meanwhile, Insert(Q, a) be the function which inserts a to Q at the end of Q, Union(Q_1, Q_2) unifies Q_1 and Q_2 by placing the content of Q_2 at the end of Q_1, and Null(Q) empties Q.

Let $v=a_1a_2...a_s \in \Delta^*$ be the task string to executed, the scheduling procedure for M can be described as:

Scheduling Procedure:

1. $Q_T:=\{a_1, a_2, ..., a_s\}$, $Q_D:=\phi$;
2. IF Q_T is empty THEN STOP;
3. u:=First(Q_T);
4. IF there exists a $t \in T(u)$ and t is enabled THEN firing t, GOTO 7;
5. IF there exists a λ-move firing sequence $e \in T_\lambda^*$ such that a $t \in T(u)$ is enabled by firing e THEN firing et, GOTO 7;
6. Insert(Q_D, u), IF Q_T is empty THEN $Q_T:=Q_D$ and Null(Q_D), GOTO 2;
7. IF Q_D is not empty THEN Q_T:=Union(Q_T, Q_D) and Null(Q_D), GOTO 2.

In the above scheduling procedure, each subtask is examined in the order which appears in the task string v. If the subtask is executable (i.e., an appropriate transition is enabled) at the time, it will be executed. Otherwise an effort is made to find a sequence of internal operations which will lead the subtask to be executable. If the effort fails again, it will be removed from the task queue Q_T to the delayed queue Q_D. Once there is a change in the state of PNT M, all the delayed subtasks in Q_D will be moved back to Q_T in their original order and be examined again, since it is desired to keep the subtask order as specified as much as possible. The scheduling procedure will terminate in a finite amount of time since it assumed that the task strings issued are compatible and complete.

There are various methods can be employed to find the λ-move sequence required by the fifth step in the scheduling procedure. For a large and complex PNT, the heuristic search algorithm discussed in [Passino and Antsaklis 1988a and b] may be used. For a bounded, average size PNT, however, the simple breadth-first search along the reachability tree of the PNT obtained by firing the only the λ-move transitions under the current marking can serve the purpose quite well.

The time factor is not explicitly considered in task scheduling. Some difficulties and optimization problems, like *starvation* and *load balancing*, may be raised when the time factor is included. In that case, methods developed for scheduling in Operation Research may be required [Shen 1988].

When applying the scheduling procedure to the dispatcher or coordinator, it should be remembered that the enabled condition of a transition is with respect to the underlying Petri net N as defined in 4.4, not to its own Petri net. In other word, the condition of the connection points have to be considered.

5.2 Task Translation

Once a subtask is located to be executed by an enabled transition during the task scheduling, the firing of the transition is actually the process of decomposing the subtask into an ordered control actions and then, after assigned with rea-time data (task formulation), executing these control actions (for the internal operation) or sending them to the corresponding unit in the next level. This phase of task execution is the task translation.

Task translation is in general very large and difficult problem in the coordination of Intelligent Machines. For a given transition, the associated task translation could be one of the major issues in the corresponding research area (e.g., pattern recognition in the vision and sensor coordinators, path planning and control algorithm in the motion coordinator, for an intelligent robotic system). The translation problem can be solved in either an active fashion or a passive fashion. In the active approaches, the translation of a subtask is formulated on-line based on a set of rules and a data base which describes the related environment and system status information. The structural formulation of plan generation suggested in [Wang and Saridis 1988b] can be used in this case. In the passive approaches, a fixed number of translations for a subtask are pre-specified and the translating is to

choose one of them according to the current situations for the subtask. The probabilistic methods are probably the most appropriate way to implement the passive task translation. Note that one of the major difference between the active and passive task translation is on the knowledge representation, the knowledge in the active task translation should be represented declaratively, and on other hand, the knowledge in the passive task translation should be represented procedurally. For a PNT, the active or passive translation is indicated by the way how the translation mapping σ is generated.

In order to avoid the discussion of task translation in too much detail, we will concentrated at the passive task translation in the following investigation. Assuming a fixed number of translations are available for each translation, we use the probabilistic method with learning ability developed by Saridis and Graham (1984) to choose the best translation for a subtask in a particular situation.

Let t be a transition of a PNT $M=(N, \Sigma, \Delta, \sigma, \mu, F)$ representing the dispatcher or a coordinator . The number of translations designed for t is

$$M_t=\sum_a |\sigma(t, a)|, \qquad \sigma(t, a) \text{ is defined for } a\in\Sigma\cup\{\lambda\}.$$

Let x_t represent the state and feedback information contained in the input places of t (x_t can be interpreted in terms of colors of tokens) and $u_t\in U_t=\{a\in\Sigma\cup\{\lambda\}|\sigma(t, a) \text{ is defined}\}$ represent the subtask to be translated by t. A *situation* is defined to be a

combination of x_t and u_t, i.e., (u_t, x_t). The number of situation distinguished by t is designated as N_t.

Now consider a matrix of subjective probabilities, $(p_{ij}^t)_{M_t \times N_t}$, such that p_{ij}^t is the subjective probability of choosing the translation s_i when the situation $(u_t, x_t)_j$ is observed. The subjective probabilities satisfy the constraint,

$$\sum_{i=1}^{M_t} p_{ij}^t = 1, \qquad \text{for all } j=1, ..., N_t.$$

The decision rule of the probabilistic method for choosing a translation is

> *Decision Rule:* When situation $(u_t, x_t)_j$ is observed, choose a translation s_i using a random strategy with the subjective probability p_{ij}^t, i=1, ..., M_t.

A random perform index or cost function is associated with each translation. After the execution of the control action specified by s_i for situation $(u_t, x_t)_j$, compute the performance index J_{ij}, and update the performance estimate using the following algorithm,

Performance Estimate Update Algorithm (PEUA):

$$\tilde{J}_{ij}(n_{ij}+1) = \tilde{J}_{ij}(n_{ij}) + \beta(n_{ij}+1)[J_{obs}(n_{ij}+1) - \tilde{J}_{ij}(n_{ij})]$$

where

J_{obs} observed performance value,

$\tilde{J}$ performance estimate, and

n_{ij} number of times the event $((u_t,x_t)_j s_i)$ has occurred.

After updating the performance estimate, update the subjective probabilities for i=1, ..., M_t and the given j by the following algorithm,

Subjective Probability Update Algorithm (SPUA):

$$p^t_{ij}(k+1)=p^t_{ij}(k) + \gamma(k+1)[\xi_{ij}(k) - p^t_{ij}(k)]$$

where

$$\xi_{ij}(k)= \begin{cases} 1, \text{ if } \tilde{J}_{ij}=\min_l \tilde{J}_{lj} \\ 0, \text{ otherwise.} \end{cases}$$

Two theorems are now stated describing the convergence properties of these algorithms.

Theorem 5.1: If the $\beta(n_{ij})$ of PEUA satisfies Dvoretsky's convergence

conditions

$$\lim_{k\to\infty}\beta(k) = 0,$$
$$\sum_{k=1}^{\infty}\beta(k) = \infty,$$
$$\sum_{k=1}^{\infty}\beta^2(k) < \infty$$

then

$$\Pr\{\lim_{k\to\infty}[\tilde{J}_{ij}(k) - \bar{J}_{ij}]=0\}=1$$

where $\bar{J}_{ij}$ is the expected value of J_{ij}.

Theorem 5.2: If the $\beta(n_{ij})$ of PEUA satisfies Dvoretsky's convergence conditions in Theorem 5.1, and the $\gamma(k)$ of SPUA satisfies Dvoretsky's convergence conditions

$$\lim_{k\to\infty}\gamma(k) = 0,$$
$$\sum_{k=1}^{\infty}\gamma(k) = \infty,$$
$$\sum_{k=1}^{\infty}\gamma^2(k) < \infty$$

then for all i and all j such that $\hat{J}_{ij} = \min_{l} \bar{J}_{lj}$

$$\Pr[\lim_{k\to\infty} p_{ij}^{t}(k) = 1] = 1$$

and for all other i and j

$$\Pr[\lim_{k\to\infty} p_{ij}^{t}(k) = 0] = 1 \ .$$

This theorem establishes the convergence of the learning algorithm under very general plant conditions. The proofs for the two theorems were given in [Saridis and Graham 1984].

Assuming that the initial performance estimates for a transition t are available, we can find *the most conservative* initial subjective probabilities by Jaynes' *Maximum Entropy Principle* [Jaynes 1968] as

$$p_{ij}^{0} = \frac{e^{-\alpha \bar{J}_{ij}^{0}}}{Z_j(\alpha)} \quad ,$$

the partition function $Z_j(\alpha)$ is defined to be

$$Z_j(\alpha) = \sum_{i=1}^{M_t} e^{-\alpha \bar{J}_{ij}^{0}} \quad ,$$

and the parameter α satisfies the constraint

$$\sum_{i=1}^{M_t} \tilde{J}_{ij}^{0} e^{-\alpha \tilde{J}_{ij}^{0}} = Z_j(\alpha)\, \tilde{J}_{j}^{0} \quad ,$$

where p_{ij}^{0} are the initial subjective probabilities, and $\tilde{J}_{ij}$ are the initial performance estimates, and $\tilde{J}_{ij}$ is the initial average performance estimate at the jth situation. When

$$\tilde{J}_{j}^{0} = \frac{\sum_{i=1}^{M_t} \tilde{J}_{ij}^{0}}{M_t}$$

we have

$$p_{ij}^{0} = \frac{\tilde{J}_{ij}^{0}}{\sum_{k=1}^{M_t} \tilde{J}_{kj}^{0}} ,$$

$i=1, \ldots, M_t$, $j=1,\ldots,N_t$.

The learning process can be measured by the entropies associated with the subjective probabilities. For a PNT M, its *translation uncertainty* is defined to be the total entropy of subjective probabilities assigned to the transitions of M, that is

$$H(M) = \sum_{t \in T} H(t)$$

and

$$H(t) = H(u_t, x_t) + H(t / u_t, x_t)$$

where

$$H(u_t,x_t) = -\sum_{j=1}^{N_t} p((u_t,x_t)_j)\ln p((u_t,x_t)_j) = -\sum_{j=1}^{N_t} p_j^t \ln p_j^t$$

where p_j^t is the objective probability of $(u_t,x_t)_j$ ocurring.

$$H(t/u_t,x_t) = -\sum_{j=1}^{N_t} p((u_t,x_t)_j)\sum_{i=1}^{M_t} p(s_i/(u_t,x_t)_j)\ln p(s_i/(u_t,x_t)_j)$$

$$= -\sum_{j=1}^{N_t} p_j^t \sum_{i=1}^{M_t} p_{ij}^t \ln p_{ij}^t$$

Define

$$H(E) = \sum_{t\in T} H(u_t, x_t), \qquad H(T/E) = \sum_{t\in T} H(t/u_t, x_t)$$

then

$$H(M) = H(E) + H(T/E)$$

This expression of H(M) indicates that the translation uncertainty can be divided into two parts: the *environment uncertainty* H(E), caused by the uncertainty of the environment (include the uncertainty in task assignment); the *pure translation uncertainty* H(T/E), the uncertainty in translation given the environment . Clearly, the learning ability of the PNT M cannot reduce the environment uncertainty H(E), but the pure translation uncertainty H(T/E) can be reduced. Note that H(E) specify the lowest bound of the total translation uncertainty, which can be achieved through the learning since the learning convergence

theorem 4.2 guarantees that the pure translation uncertainty can be reduced to zero by learning process.

The *average execution cost estimate* of the PNT M is

$$\tilde{J}(M)=\sum_{t\in T}\tilde{J}(t)$$

where

$$\tilde{J}(t)=\sum_{j=1}^{N_t}p_j^t\sum_{i=1}^{M_t}p_{ij}^t\tilde{J}_{ij}$$

For the whole coordination structure, we have

$$H(CS) = H(E_{CS}) + H(T_{CS})$$

where

$$H(E_{CS})=H(E_D) + \sum_{i=1}^{N} H(E_{C_i})$$

$$H(T_{CS})=H(T_D) + \sum_{i=1}^{N} H(T_{C_i})$$

The decision making mechanism of the whole coordination structure bears the characteristics of the *embedded decision schema* defined in [Saridis and Graham 1984]. The learning process of the whole coordination structure also has the similar structure as

that of the *hierarchical learning automata* [Mandyam et al 1981, Baba 1987]. For the case of the small number of situations, the learning by recording the conditional probabilities under specific situations will not cause the serious problem in memory space and learning speed. As the number of situations and the task to be processed increasing, however, the problem of memory space and learning speed becomes more and more serious. For the case of the larger number of situations, the pattern-recognizing learning algorithm developed in [Barto and Anandan 1985], which avoids the maintenance of separate selection probabilities for each situation by parameterizing the conditional probabilities and constructing a mapping from the situations to the parameter, should be used.

A note is in order concerning the case when the pure translation uncertainty of a PNT is reduced to zero. In this case, the optimal translation is founded for all transitions in the PNT. However, this does not necessarily imply that a global optimal translation is achieved for the PNT in general, simply because that the optimal translations of a transition might cause the worse situations for the subsequent transitions, thus increase the total task execution cost. To achieve the global optimal translation, it is imperative to specify the influences of the translation of a transition to others, and use a learning algorithm based the global information. However, an analytical expression for such influences may be too complex to be established for transitions with quite different functions. If some appropriate forms for such influences are obtained, a team-theoretic formulation for the global optimal translation might be developed.

A PNT with the smaller pure translation uncertainty indicates the PNT is more knowledgeable in task execution. Since learning from the execution can reduce the pure translation uncertainty, learning can make a PNT more knowledgeable about task execution. This observation reveals that the decision-makings in the Organization Level and Coordination Level bear "dual" character: on one side, the decisions made in Organization Level have to make the dispatcher to accomplish the requested task with some optimization criterion, on the other side, these decisions should also make the dispatcher more knowledgeable about the task execution in the future by reducing its pure translation uncertainty. Similarly, the decisions made in the dispatcher have to make the coordinators to accomplish the requested task with some optimization criterion, on the other side, these decisions should also make the coordinators more knowledgeable about the task execution in the future by reducing their pure translation uncertainty. To reduce the cost, only the subtasks with the minimum costs should be selected; To reduce the uncertainty, the subtasks with the large entropy should be tried. The trade-off of the two sides should be judged by some criterion.

6. A Case Study

In this chapter we present a case study for an Intelligent Manipulator System by building a simple coordination structure for the Coordination Level consisting of a dispatcher, a vision coordinator, a sensor coordinator, and a motion coordinator, and simulating the task process on the model. The investigation here serves only for the illustrative purpose and the model constructed does not exhibit fully the real expression power of coordination structure, since both the dispatcher and the coordinators are oversimplified and concurrency is not involved. The first section of the chapter gives the PNT models for the dispatcher and the coordinators, and the overall structure of the Coordination Level. The second section shows the simulation results of the task processes for the corresponding PNT models and the whole coordination structure.

6.1 The Petri Net Transducers and the Coordination Structure

This section describes the individual PNTs for the dispatcher and coordinators at first, and then integrates the PNTs by specifying the corresponding receiving and sending facilities to form the final simple coordination structure for the Coordination Level. We start from the dispatcher.

6.1.1 The Petri Net Transducer for the Dispatcher

A. Petri net Model: The Petri net model for the dispatcher, given in Figure 6.1, consists of 8 places and 10 transitions. A transition generally represents an algorithm for some specific task and a place generally represents some specific process. For the dispatcher, these places and transitions are specified as follows:

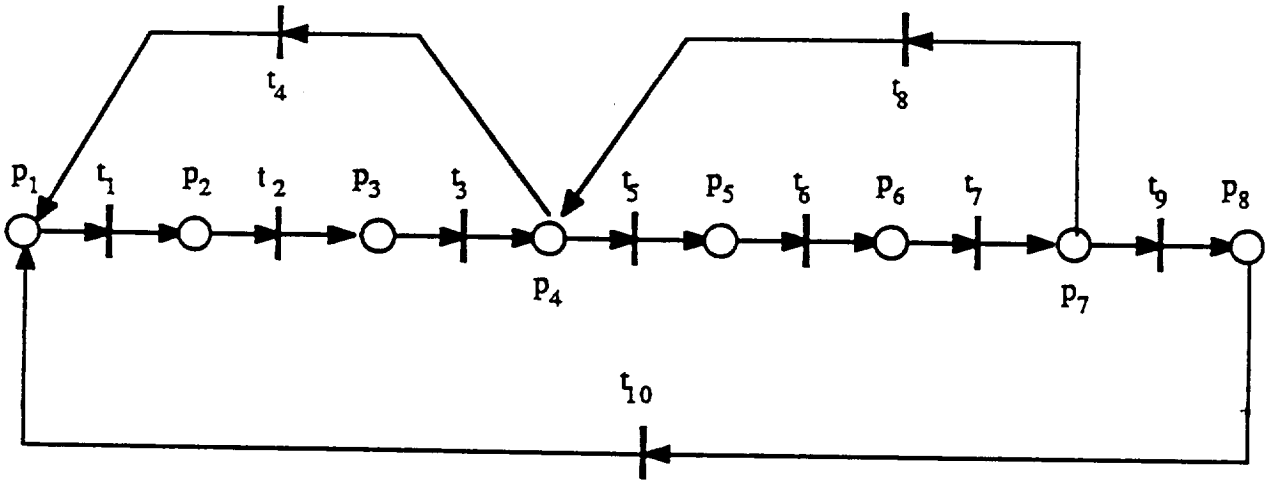

Fig. 6.1 The Petri Net for The Dispatcher

Places:

p_1: initialization process;

p_2: gross environment sensing process;

p_3: arm motion process;

p_4: arm motion verification: check whether or not the desired location of upper arm is achieved;

p_5: fine environment sensing process;

p_6: hand motion process process;

p_7: hand motion verification: check whether or not the specified object is grasped or putted down on the desired location ;

p_8: task verification: check whether or not the specified task is completed.

Transitions:

t_1: issuing commands for the gross environment sensing;

t_2: receiving the gross environment sensing information from the coordinators and issuing commands for upper arm motion;

t_3: receiving and verifying the motion execution result from the coordinators;

t_4: returning to the initial place p_1 because the failure of the motion process;

t_5: issuing commands for the fine environment sensing;

t_6: receiving the the fine environment information from the coordinators and issuing commands for hand motion;

t_7: receiving and verifying the hand motion execution result from the coordinators;

t_8: returning to the arm motion verification place p_4 because the failure of the fine motion process;

t_9: starting the task verification;

t_{10}: continuing the task process.

B. Task Translation: The input alphabet, i.e., the set of the primitive events from the Organization Level, is assumed to be $\Sigma_o=\{e_1, e_2, e_3, e_4, e_5\}$, where

e_1=Dump, e_2=Grasp, e_3=Fill, e_4=Move, e_5=Camera

are defined based on the example given in [Valavanis 1986].

The output alphabet, i.e., the set of the coordinator-oriented control actions for the coordinators is $\Delta_O= \Sigma_C= \Delta_I \cup \Sigma_v \cup \Sigma_s \cup \Sigma_m$, where Δ_I is the set of the internal operations of the dispatcher, and Σ_v, Σ_s, Σ_m are the input alphabets for the vision coordinator, the sensor coordinator, and the motion coordinator, respectively. We assume that

$$\Delta_I=\{i_1, i_2, i_3, i_4, i_5, i_6\},$$

$$\Sigma_v=\{v_1, v_2\},\ \Sigma_s=\{s_1, s_2\},\ \Sigma_v=\{m_1, m_2, h_1, h_2\}$$

where

i_1=the procedure for verifying the arm motion execution,

i_2=the procedure for preparation of re-executing the arm motion,

i_3=the procedure for verifying the hand motion execution,

i_4=the procedure for preparation of re-executing the hand motion,

i_5=the procedure for organizing the feedback for the Organization Level,

i_6=the procedure for continuing the task execution;

and the v_i's, s_i's, as well as m_i' and h_i's are instructions to the vision, the sensor, and the motion coordinators, respectively, and will be defined in the descriptions for the corresponding coordinators.

The translation mapping σ_d for the dispatcher is specified as

$$\sigma_d(t_6, e_1)=\sigma_d(t_6, e_2)=\sigma_d(t_6, e_3)=\{h_1s_1, h_1s_2, h_2s_1, h_2s_2, h_1, h_2\},$$

$$\sigma_d(t_2, e_4)=\sigma_d(t_2, \lambda)=\{m_1v_1, m_1v_2, m_2v_1, m_2v_2, m_1, m_2\},$$

$$\sigma_d(t_1, e_5)=\sigma_d(t_1, \lambda)=\{v_1, v_2\},$$

$\sigma_d(t_5, e_5)=\sigma_d(t_5, \lambda)=\{v_1s_1, v_2s_1, v_1s_2, v_2s_2, s_1, s_2\}$,

$\sigma_d(t_3, \lambda)=\{i_1\}$, $\sigma_d(t_4, \lambda)=\{i_2\}$, $\sigma_d(t_7, \lambda)=\{i_3\}$,

$\sigma_d(t_8, \lambda)=\{i_4\}$, $\sigma_d(t_9, \lambda)=\{i_5\}$, $\sigma_d(t_{10}, \lambda)=\{i_6\}$.

C. Operation procedure: Once receiving a task plan from the Organization Level, the dispatcher will issue one of the two control strings v_1 and v_2, according to the information coded in the tokens of the initial place p_1 and the task plan, to the vision coordinator by firing transition t_1. Upon the completion of the vision process, there will be a token in output point of the vision coordinator, which contains the gross information about the environment. Now the transition t_2 is enabled, and the motion coordinator is activated through the firing of transition t_2 to move the upper arm to the designed position. Transition t_2 selects the control command among $\{m_1v_1, m_1v_2, m_2v_1, m_2v_2, m_1, m_2\}$ based on the information contained in its input tokens. The first four control strings of t_2 need the vision information for the execution verification, and the last two only use the feedback from the motion coordinator. When the arm motion process is completed, according to the result of the execution verification carried out by t_3, either the fine sensing transition t_5 is fired, if the arm motion is successful, or transition t_4 is fired to re-execute the arm moving task, if the arm motion is fail (by a certain criterion). Transition t_5 can pick up one of the control commands among $\{v_1s_1, v_2s_1, v_1s_2, v_2s_2, s_1, s_2\}$. The first four control strings of t_5 involved the cooperation between the vision coordinator and the sensor coordinator, however, the last two only use the sensor coordinator. After the completion of the fine sensing process, transition t_6 can be fired to invoke the motion coordinator for hand motion , e.g., to grasp or put down, to open or close some thing. Like t_2, transition t_6 also has six alternatives,$\{h_1s_1, h_1s_2, h_2s_1, h_2s_2, h_1, h_2\}$, for its task execution, with the first four need the sensor information for the execution verification, and the last two only use the feedback from the motion coordinator. The result of the hand motion verification, processed by transition t_7, will enable the next two transitions, t_8 and t_9. The successful

process leads to the firing of t_9, otherwise, of t_8, which may further lead to the firing of t_4 if the desired hand manipulation can not be achieved by the current upper arm location. Finally, the task verification transition t_9 will decide whether or not the whole task plan is completed. If the task plan is achieved, the dispatching process will be ended up by forming the execution feedback required the Organization Level, otherwise, transition t_{10} will be fired to continue to the execution process for the remained subtasks.

6.1.2 The Petri Net Transducer for the Vision Coordinator

A. Petri net Model: The Petri net model for the vision coordinator, given in Figure 6.2, consists of 5 places and 9 transitions. As for the dispatcher, a transition generally represents an algorithm for some specific task and a place generally represents some specific process. For the vision coordinator, these places and transitions are specified as follows:

Places:

p_s: the start place;

p_1: initialization process;

p_2: feature extraction and identification process;

p_3: image fusion process;

p_f: the final place.

Transitions:

t_s: initializing the vision system;

t_1: calibrating;

t_2: taking picture and performing feature extraction and identification;

t_3: returning to the initial place p_1 because the failure of feature extraction and identification;

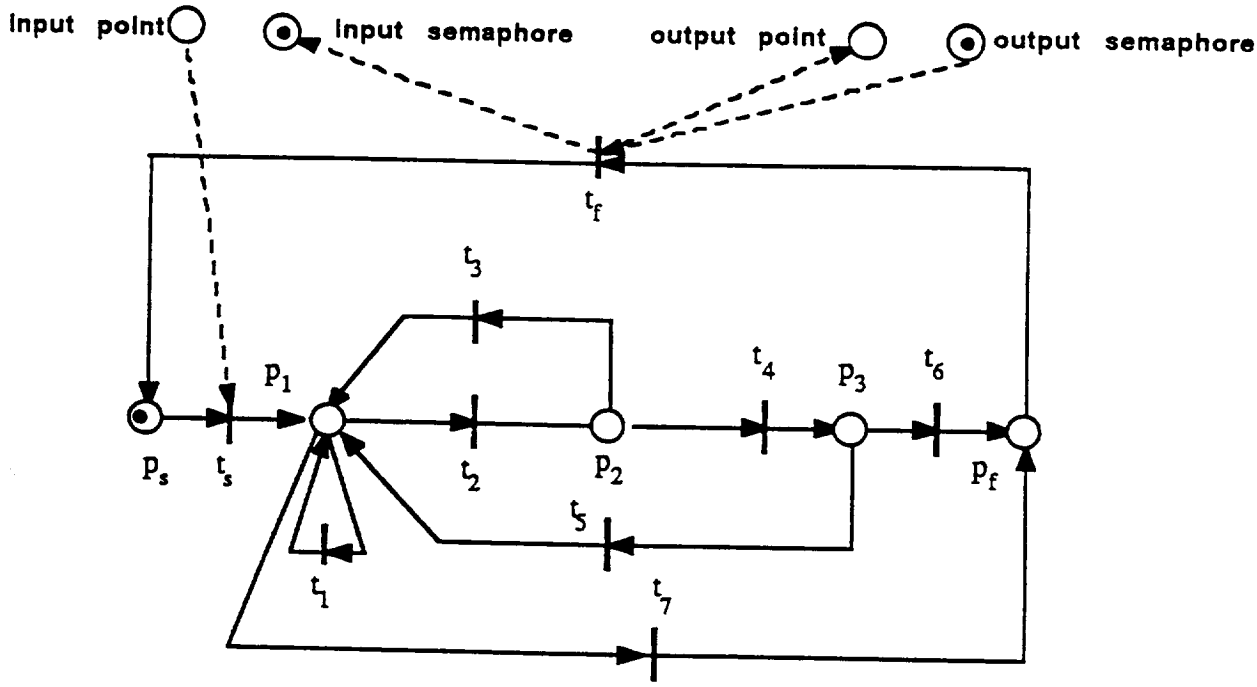

Fig 6.2. The Petri Net for the Vision Coordinator

t_4: performing the image fusion to obtain the required information, like location and orientation;

t_5: returning to the initial place p_1 because the failure of the image fusion;

t_6: finishing the vision process;

t_7: acknowledging the failure of the vision process;

t_f: reporting the visual information and resetting the vision input semaphore and the start place.

B. Task Translation: The input alphabet is assumed to be $\Sigma_v=\{v_1, v_2\}$, where

v_1=the minimum-error vision procedure,

v_2=the minimum-time vision procedure.

The output alphabet Δ_v of the vision coordinator consists of the internal procedures and the hardware-oriented operations for the devices associated with it. Since we are not going to involve with the Execution Level, we will not specified Δ_v here, but point out that only transitions t_1 and t_2 pass operation instructions to the camera controllers and all other transitions just deal with internal information processing, e.g., feature extraction, pattern recognition, and image fusion. By the same reason, the translation mapping σ_v for the vision coordinator will not be fully specified. However, in order to perform the simulation, for transitions t_2 and t_4 , we assume that

$$\sigma_v(t_2, \lambda)=\{vg_1, vg_2\},$$

$$\sigma_v(t_4, \lambda)=\{f_1, f_2\},$$

where

v =the operation instructions for the camera controllers,

g_1=the fast algorithm for feature extraction and pattern identification,

g_2=the accurate algorithm for feature extraction and pattern identification,

f_1=the fast algorithm for image fusion,

f_2=the accurate algorithm for image fusion.

C. Operation procedure: Once receiving a task from the dispatcher (when a token is displaced in its input point, specifying the task is either v_1 or v_2), the vision coordinator starts the process by firing t_s. Depending on the information coded in the token of the initial place p_1 and the current system status, the coordinator starts either transition t_1 or t_2, i.e., either to set up camera angle, focus, lens, etc., and to calibrate the cameras first, and then take picture, or to take picture without calibrating. Transition t_2 can select one of the two operations vg_2 and vg_2 according to the information coded in its input token and the task issued. If the relevant features and patterns are extracted and identified in the picture obtained, then transition t_4 is fired to perform the image fusion process using the information collected by t_2, otherwise, t_3 is fired to back to p_1 to take new picture. Transition t_4 also has two alternatives for image fusing, f_1 and f_2. Upon the completion of fusion process, either transition t_6 or transition t_5 will be fired. A successful fusion process, i.e., the required visual information is obtained, results in the end of the vision process (the firing of t_6), otherwise leads to the firing of t_5, which may further lead to the firing of t_7 if it is decided that the desired information can not be obtained by the vision system from the environment. Once a token reaches the final place p_f, the final transition t_f will be fired to report the result to the dispatcher and to reset the input semaphore and the start place, indicating that the vision coordinator is available for task again.

6.1.3 The Petri Net Transducer for the Sensor Coordinator

A. Petri net Model: The Petri net model for the sensor coordinator has the same structure as that of the PNT for the vision coordinator, as shown in Figure 6.3, but with different transition and place function. Places and transitions for the sensor coordinator are specified as follows:

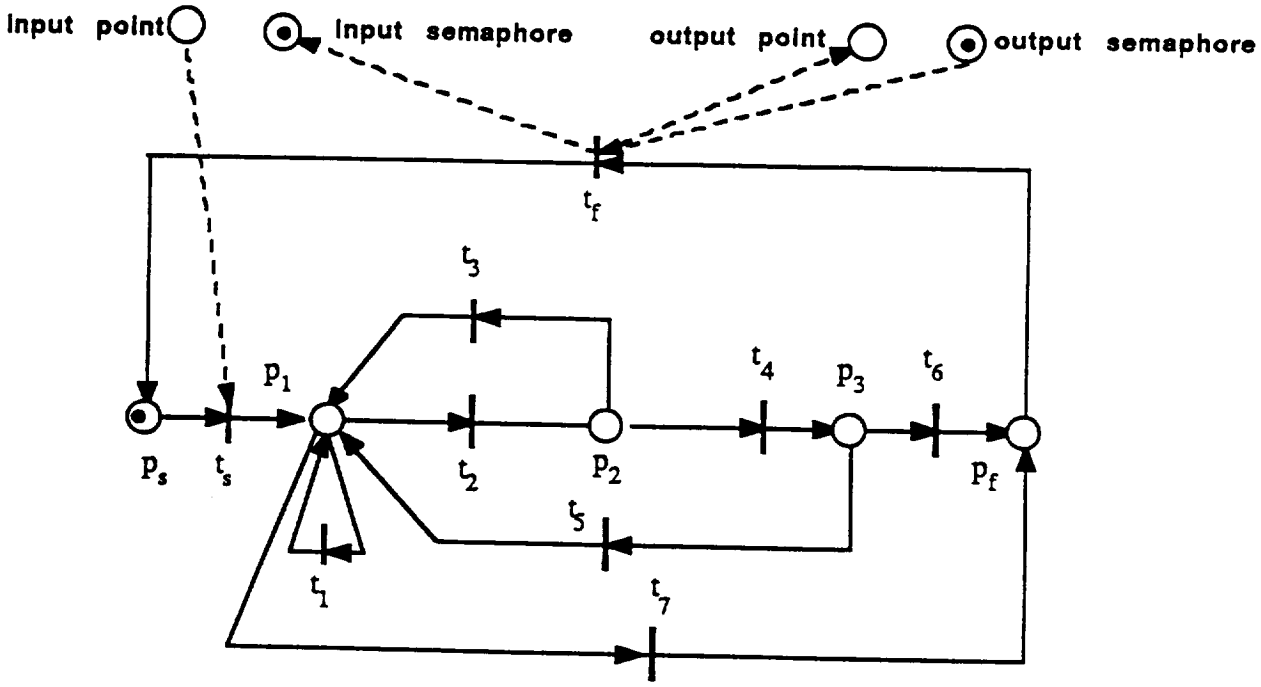

Fig 6.3. The Petri Net for the Sensor Coordinator

Places:

p_s: the start place;

p_1: initialization process;

p_2: feature extraction and identification process;

p_3: feature fusion process;

p_f: the final place.

Transitions:

t_s: initializing the sensor system;

t_1: calibrating;

t_2: taking data and performing feature extraction and identification;

t_3: returning to the initial place p_1 because the failure of feature extraction and identification;

t_4: performing the feature fusion to obtain the required information, like location and orientation;

t_5: returning to the initial place p_1 because the failure of the image fusion;

t_6: finishing the sensor process;

t_7: acknowledging the failure of the sensor process;

t_f: reporting the results and resetting the sensor input semaphore.

B. Task Translation: The input alphabet is assumed to be $\Sigma_s=\{s_1, s_2\}$, where

s_1=the minimum-error sensor procedure,

s_2=the minimum-time sensor procedure.

The output alphabet Δ_s of the sensor coordinator, as for the vision coordinator, will not be specified here. Similar to the vision coordinator, only transitions t_1 and t_2 pass

operation instructions to the sensor controllers and all other transitions just deal with internal information processing. Again, for transitions t_2 and t_4 , we assume that

$$\sigma_s(t_2, \lambda)=\{sd_1, sd_2\},$$

$$\sigma_s(t_4, \lambda)=\{c_1, c_2\},$$

where

s =the operation instructions for the sensor controllers,

d_1=the fast algorithm for feature extraction and pattern identification,

d_2=the accurate algorithm for feature extraction and pattern identification,

c_1=the fast algorithm for feature fusion,

c_2=the accurate algorithm for feature fusion.

C. Operation procedure: Once receiving a task from the dispatcher (when a token is displaced in its input point, specifying the task is either s_1 or s_2), the sensor coordinator starts the process by firing t_s. Depending on the information coded in the token of the initial place p_1 and the current system status, the coordinator starts either transition t_1 or t_2, i.e., either to set up sensor parameter and to calibrate the sensors first, and then take data, or to take data without calibrating. Transition t_2 can select one of the two operations sd_2 and sd_2 according to the information coded in its input token and the task issued. If the relevant features and patterns are extracted and identified in the data obtained, then transition t_4 is fired to perform the feature fusion process using the information collected by t_2, otherwise, t_3 is fired to back to p_1 to take new data. Transition t_4 also has two alternatives for feature fusing, c_1 and c_2. Upon the completion of fusion process, either transition t_6 or transition t_5 will be fired. A successful fusion process, i.e., the required information is obtained, results in the end of the sensor process (the firing of t_6), otherwise leads to the firing of t_5, which may further lead to the firing of t_7 if it is decided that the desired information can not be obtained by the sensor system from the environment. Once a token reaches the final place p_f, the final transition t_f will be fired to report the result to the dispatcher and to reset

the input semaphore and the start place, indicating that the sensor coordinator is available for task again.

6.1.4 The Petri Net Transducer for the Motion Coordinator

A. Petri net Model: The Petri net model for the motion coordinator, given in Figure 6.4, consists of 7 places and 12 transitions. These places and transitions are specified as follows:

Places:

p_s: the start place;

p_1: initialization process;

p_2: path planning process;

p_3: arm moving process;

p_4: hand approaching process;

p_5: grasping approaching process;

p_f: the final place.

Transitions:

t_s: initializing the motion system;

t_1: calibrating;

t_2: planning path for arm;

t_3: issuing arm motion instructions;

t_4: returning to the initial place p_1 because the desired location is not reached;

t_5: finishing the arm motion;

t_6: determining the path for hand to approach the desired object (fine path planning), and issuing hand approach instructions;

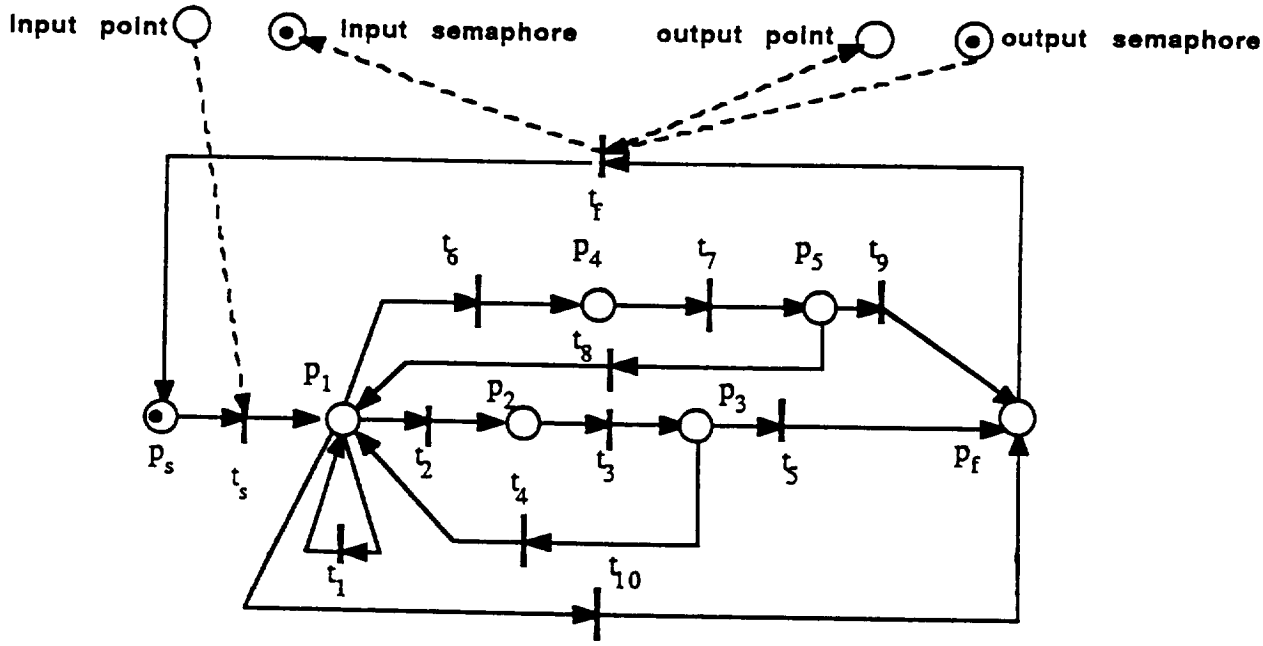

Fig 6.4. The Petri Net for The Motion Coordinator

t_7: issuing grasping instructions;

t_8: returning to the initial place p_1 because the specified object is not seized or not putted to the desired location;

t_9: finishing grasping process;

t_{10}: acknowledging the failure of motion process;

t_f: reporting execution results and resetting the motion input semaphore and the start place.

B. Task Translation: The input alphabet is assumed to be $\Sigma_m=\{m_1, m_2, h_1, h_2\}$, where

m_1=the minimum-error motion procedure,

m_2=the minimum-time motion procedure,

h_1=the minimum-error grasp procedure,

h_2=the minimum-time grasp procedure.

As for the vision coordinator and the sensor coordinator, the output alphabet Δ_m of the motion coordinator is not specified here. The transitions which pass operation instructions to the arm or hand controllers are t_1, t_3, t_6, and t_7. All other transitions deal with internal information processing like path planing and execution verification. The translation mapping σ_v for transitions t_3 and t_7 are assumed to be

$\sigma_m(t_3, \lambda)=\{l_1, l_2\}$,

$\sigma_m(t_7, \lambda)=\{k_1, k_2\}$,

where

v =the operation instructions for the camera controllers,

l_1=the fast arm control algorithm for arm controllers,

l_2=the accurate arm control algorithm for arm controllers,

k_1=the fast hand control algorithm for hand controllers,

k_2=the accurate hand control algorithm for hand controllers,

C. Operation procedure: Once receiving a task from the dispatcher (when a token is displaced in its input point, specifying the task is which of $\{m_1, m_2, h_1, h_2\}$), the motion coordinator starts the process by firing t_s. Depending on the information coded in the token of the initial place p_1 and the current system status, the coordinator starts either transition t_1 or one of t_2 and t_6, i.e., either to calibrate the arm or hand first, and then to execute arm or hand motions, or to execute arm or hand motions without calibrating. If the task is to move the upper arm (i.e., one of m_1 and m_2 is received), transition t_2 will be fired, otherwise the task is to move the hand (i.e., one of h_1 and h_2 is received) and t_6 will be fired. Both transitions t_3 and t_7 have two alternatives to control the arm (l_1 and l_2) or the hand (k_1 and k_2). The successful execution of the arm motion (by t_3) or hand grasp (by t_7) will lead to the firing of transitions t_5 or t_9, indicating the end of the successful motion process, otherwise transitions t_4 or t_8 will be fired, which may further result in the firing of t_{10}, acknowledging the failure of the motion process. In any case, once a token reaches the final place p_f, the final transition t_f will be fired to report the execution result to the dispatcher and to reset the input semaphore and the start place, indicating that the motion coordinator is available for task again.

6.1.5 The Simple Coordination Structure for the Coordination Level

The (simple) coordination structure for the Coordination Level now can be constructed from the individual PNTs for the dispatcher and coordinators by introducing the connection points and the receiving and sending functions. The pattern of the connection points in a coordination structure is pre-fixed, i.e., each coordinator is associated with four connection points, the input point, the input semaphore, the output point, and the output semaphore.

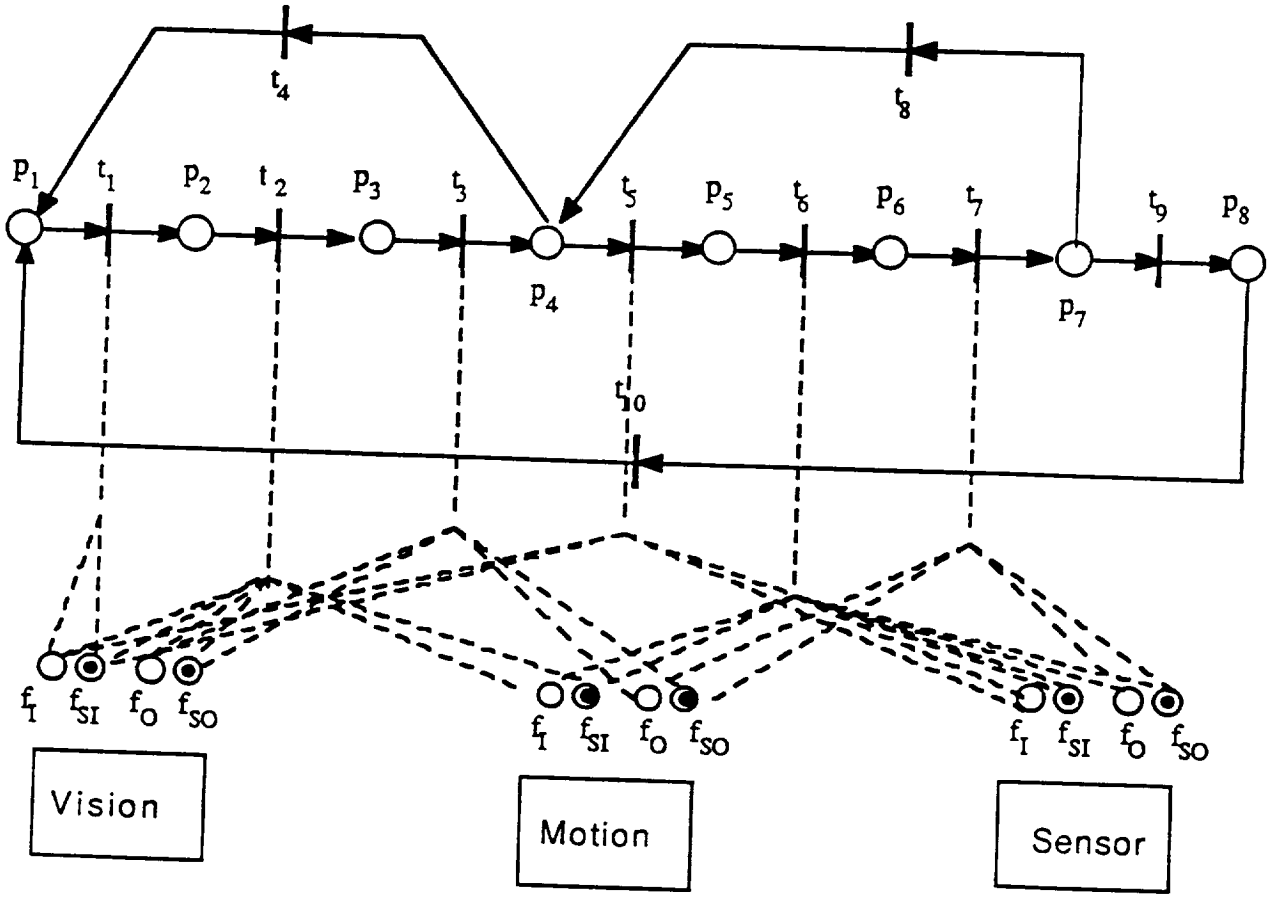

Fig. 6.5 The Simple Coordination Structure

The dispatcher receiving and sending functions are specified as

$$R_D=\{(t_1, f^v_{SI}), (t_2, f^v_{SI}), (t_2, f^v_O), (t_2,f^m_{SI}), (t_3, f^v_O), (t_3, f^m_O), (t_5, f^v_{SI}), (t_5, f^s_{SI}),$$
$$(t_6, f^s_{SI}), (t_6, f^v_O), (t_6, f^s_O),(t_6,f^m_{SI}), (t_7, f^s_O), (t_7, f^m_O)\}$$

$$S_D=\{(t_1, f^v_I), (t_2, f^v_I), (t_2, f^v_{SO}), (t_2,f^m_I), (t_3, f^v_{SO}),, (t_3, f^m_{SO}), (t_5, f^v_I), (t_5, f^s_I),$$
$$(t_6, f^s_I), (t_6, f^v_{SO}), (t_6, f^s_{SO}),(t_6,f^m_I), (t_7, f^s_{SO}), (t_7, f^m_{SO})\}$$

The coordinator receiving and sending functions for a simple coordination structure are fixed. In our case, it follows that for each coordinator

$$\{(t_s, f_I), (t_f, f_{SO})\}\in R_C, \qquad \{(t_f, f_{SI}), (t_f, f_O)\}\in S_C$$

as shown by the dash arcs in the Figures 6.2-4.

Figure 6.5 gives the overall structure of the coordination structure (the arrows for the receiving and sending links are omitted). It should be pointed out that the PNs of the dispatcher and coordinators are bounded and live, therefore the coordination structure is also bounded and live by the theorems 4.3 and 4.4.

6.2 Simulation of the Task Processes in the Coordination Structure

Now we describe the results of the task processing simulation on the coordination structure constructed above.

A. Assumptions: From the section 5.2, to simulate the task processing, we have to (i) specify the situations for each transition in the dispatcher and the coordinators, (ii) assume the cost functions (performance indices) for each task translation of a transition.

The situations can be specified in terms of the information encoded in the tokens of a Petri net (or colors of tokens, as called in the colored Petri nets). For the dispatcher, we assume that the tokens in the places can be classified into one of the two types: the *normal* token and the *special* token. The meaning of "normal" and "special" can vary from place to place. Besides the normal and the special types, the tokens in the places of p_4 and p_7 may take another type: the *back* token, and the tokens in the places of p_8 may be a *finish* type. This can be written as

$$p_1=p_2=p_3=p_5=p_6=\{n, s\},\ p_4=p_7=\{n, s, b\},\ p_8=\{n, s, f\},$$

When the token in p_4 or p_7 is the back type, the transition t_4 or t_8 has to be fired. For p_8, a finish token indicates the completion of the task execution, otherwise, the transition t_{10} has to be fired to continue the task processing. We assumed that for every places, the normal tokens are desired.

For the connection points, the tokens in the output points are assumed to have only two types: the *successful* token indicating the task has been well executed and the *fail* token indicating the task has been poorly executed. The meaning of "successful" and "fail" depends the particular criterion selected (i.e., the "fail" do not necessarily means the failure of the execution) and can vary from coordinator to coordinator. However, we assumed that successful tokens are more desired than the fail tokens.

The situations for the transitions of the dispatcher now can be summarized as

t_1: $\{n, s\}^1$, $M_1=2$;

t_2: $\{n, s\}^2 \times \{succ, fail\}^v$, $M_2=4$;

t_3: $\{n, s\}^3 \times \{succ, fail\}^v \times \{succ, fail\}^m$, $M_3=8$;

t_4: $\{b\}^4$, $M_4=1$;

t_5: $\{n, s\}^4$, $M_5=2$;

t_6: $\{n, s\}^5 \times \{succ, fail\}^v \times \{succ, fail\}^s$, $M_6=8$;

t_7: $\{n, s\}^5 \times \{succ, fail\}^v \times \{succ, fail\}^m$, $M_7=8$;

t_8: $\{b\}^7$, $M_8=1$;

t_9: $\{n, s\}^7$, $M_9=2$;

t_{10}: $\{n, s\}^8$, $M_{10}=2$;

superscripts of { } indicate the places that the tokens in { } belong to. Note that, to simplify the simulation, we assume that the transition t_7 treats the three subtasks e_1, e_2, and e_2 as the same, otherwise, t_7 will have 3×8=24 different situations. To further simplify the problem, it is assumed that the transitions t_3, t_6, and t_7 classified the feedbacks from the coordinators into two classes:

t_3: $c_1=\{(suss^v, suss^m), (fail^v, succ^m), suss^m\}$, $c_2=\{(succ^v, fail^m), (fail^v, fail^m), fail^m\}$

t_6: $c_1=\{(suss^v, suss^s), (fail^v, succ^s), suss^s\}$, $c_2=\{(succ^v, fail^s), (fail^v, fail^s), fail^s\}$

t_7: $c_1=\{(suss^s, suss^m), (fail^s, suss^m), suss^m\}$, $c_2=\{(suss^s, fail^m), (fail^s, fail^m), fail^m\}$

where the superscript of succ or fail indicates where the feedback comes from. Therefore the numbers of the situations for the transitions t_3, t_6, and t_7 are reduced to $M_3=M_6=M_7=4$.

For the vision coordinator, we only consider the translations in the transitions t_2 and t_4. The executions by t_2 and t_4 are also assumed to be always successful, thus the transitions t_3, t_4, and t_7 will be never be fired. The tokens in the place p_2 (or the execution results of t_2) can be y_1 or y_2, and the execution result of t_4 is either "successful" or "fail", as already assumed for the output points. The situations for t_2 and t_4 are assumed to be

t_2: $\{v_1, v_2\}$, $M_2=2$;

t_4: $\{v_1, v_2\} \times \{y_1, y_2\}^2$, $M_4=4$;

The consideration for the sensor coordinator is the same as that for the vision coordinator, that is, the executions by t_2 and t_4 are assumed to be always successful. The tokens in the place p_2 can be z_1 or z_2, and the execution result of t_4 is either "successful" or "fail". The situations for t_2 and t_4 are assumed to be

t_2: $\{s_1, s_2\}$, $M_2=2$;

t_4: $\{s_1, s_2\} \times \{z_1, z_2\}^2$, $M_4=4$;

For the motion coordinator, only the translations in the transitions t_3 and t_7 are considered. Similarly, the executions by t_3 and t_7 are assumed to be always successful, thus the transitions t_4, t_8, and t_{10} will be never be fired. The execution result of t_3 or t_7 is either "successful" or "fail", as already assumed for the output points. The situations for t_3 and t_7 are assumed to be

t_3: $\{m_1, m_2\}$, $M_3=2$;

t_7: $\{h_1, h_2\}$, $M_7=4$;

The cost functions for the transitions in the dispatcher are assumed to be

	$(p_2=n, f_O^v=succ)$	$(p_2=n, f_O^v=fail)$	$(p_2=s, f_O^v=succ)$	$(p_2=s, f_O^v=fail)$
$J(t_1)=$	1	2	2	5
	$(p_3=n, f_O=c_1)$	$(p_3=n, f_O=c_2)$	$(p_3=s, f_O=c_1)$	$(p_3=s, f_O=c_2)$
$J(t_2)=$	1	2	2	5
	$(p_5=n, f_O=succ)$	$(p_5=n, f_O=fail)$	$(p_5=s, f_O=succ)$	$(p_5=s, f_O=fail)$
$J(t_5)=$	1	2	2	5
	$(p_6=n, f_O=c_1)$	$(p_6=n, f_O=c_2)$	$(p_6=s, f_O=c_1)$	$(p_6=s, f_O=c_2)$
$J(t_6)=$	1	2	2	5

that is, when a normal token in the place p_2 and a successful token in the vision output f_O^v are observed after the execution of t_1, the cost is 1; and when a normal token in the place p_3 and a class 2 feedback from the vision output f_O^v and the vision output f_O^m are observed after

the execution of t_2, the cost is 2; ..., etc. To model the effects of the transition executions, we assume the following conditional probabilities

t_1: prob(p_2=n |v_1, p_1=n)=4/5; prob(p_2=n |v_2, p_1=n)=2/5;
prob(p_2=n |v_1, p_1=s)=2/5; prob(p_2=n |v_2, p_1=n)=1/5;
prob(p_2=s |•) can determined by prob(p_2=s |•)=1 - prob(p_2=n |•).

t_5: prob(p_5=n |u_1, p_4=n)=4/5; prob(p_5=n |u_2, p_4=n)=2/5;
prob(p_5=n |u_1, p_4=s)=2/5; prob(p_5=n |u_2, p_4=n)=1/5;
prob(p_5=s |•) can determined by prob(p_5=s |•)=1 - prob(p_5=n |•).

t_2: prob(p_3=n |u_1, n_1)=4/5; prob(p_3=n |u_2, n_1)=3/5; prob(p_3=n |u_3, n_1)=2/5;
prob(p_3=n |u_1, n_2)=3/5; prob(p_3=n |u_2, n_2)=2/5; prob(p_3=n |u_3, n_2)=1/5;
prob(p_3=n|u_1,n_3)=3/10; prob(p_3=n |u_2, n_3)=1/5; prob(p_3=n |u_3, n_3)=1/10;
prob(p_3=s |•) can determined by prob(p_3=s |•)=1 - prob(p_3=n |•).

t_6: prob(p_6=n |u_1, n_1)=4/5; prob(p_6=n |u_2, n_1)=3/5; prob(p_6=n |u_3, n_1)=2/5;
prob(p_6=n |u_1, n_2)=3/5; prob(p_6=n |u_2, n_2)=2/5; prob(p_6=n |u_3, n_2)=1/5;
prob(p_6=n|u_1,n_3)=3/10; prob(p_6=n |u_2, n_3)=1/5; prob(p_6=n |u_3, n_3)=1/10;
prob(p_6=s |•) can determined by prob(p_6=s |•)=1 - prob(p_6=n |•).

To simplify the simulation, the translation strings for t_5 has been grouped into two sets u_1 and u_2, and the translation strings for t_2 or t_6 has been grouped into three sets u_1, u_2, and u_3. The four situations for t_2 or t_6 are also reduced to three cases n_1, n_2, and n_3, according to the assumed cost functions for two transitions. The reductions are specified as

t_5: $u_1=\{v_1s_1, v_2s_1, s_1\}$, $u_2=\{v_1s_2, v_2s_2, s_2\}$;

t_2: $u_1=\{m_1v_1\}$, $u_2=\{m_1v_2, m_2v_1, m_1\}$, $u_3=\{m_2v_2, m_2\}$;
$n_1=\{(n, c_1)\}$, $n_2=\{(n, c_2), (s, c_1)\}$, $n_3=\{(s, c_2)\}$;

t_6: $u_1=\{h_1s_1\}$, $u_2=\{h_1s_2, h_2s_1, h_1\}$, $u_3=\{h_2s_2, h_2\}$;

$n_1=\{(n, c_1)\}$, $n_2=\{(n, c_2), (s, c_1)\}$, $n_3=\{(s, c_2)\}$;

No learning is performed for the internal transitions t_3, t_4, t_7, t_8, t_9, and t_{10}. The decision-makings for them are assumed either deterministic or pure random ones, as given in the following,

t_3: $\text{prob}(p_4=n|p_3=n,f_O=c_1)=1$, $\text{prob}(p_4=s|p_3=n,f_O=c_2 \text{ or } p_3=s,f_O=c_1)=1$, $\text{prob}(p_4=b|p_3=s,f_O=c_2)=1$;

t_4: $\text{prob}(p_1=n|p_4=b) = \text{prob}(p_1=s|p_4=b) = 1/2$;

t_8: $\text{prob}(p_4=n|p_7=b) = \text{prob}(p_4=s|p_7=b) = \text{prob}(p_4=b|p_7=b) = 1/3$;

t_9: $\text{prob}(p_8=n|p_7=n) = \text{prob}(p_8=s|p_7=s) = 1$ if the task has not been completed;

t_{10}: $\text{prob}(p_1=n|p_8=n) = \text{prob}(p_1=s|p_8=s) = 1$;

for the transition t_9, the completion of the task means no subtasks are left on the input tape of the dispatcher.

The cost functions for the transitions in the coordinators are assumed to be

vision: $J(t_2)=1$ if $p_2=y_2$, $J(t_2)=0$ if $p_2=y_1$;
$J(t_4)=1$ if $p_3=\text{succ}$, $J(t_4)=0$ if $p_3=\text{fail}$;

sensor: $J(t_2)=1$ if $p_2=z_2$, $J(t_2)=0$ if $p_2=z_1$;
$J(t_4)=1$ if $p_3=\text{succ}$, $J(t_4)=0$ if $p_3=\text{fail}$;

motion: $J(t_3)=1$ if $p_3=\text{succ}$, $J(t_3)=0$ if $p_3=\text{fail}$,
$J(t_7)=1$ if $p_5=\text{succ}$, $J(t_7)=0$ if $p_5=\text{fail}$;

clearly, we assume the execution result y_1 or z_1 is more desired that y_2 or z_2.

The effects of the transition executions are assumed as:

vision: $prob(p_2=y_1|vg_1, v_1) = 4/5$, $prob(p_2=y_1|vg_2, v_1) = 1/5$,
$prob(p_2=y_1|vg_1, v_2) = 1/5$, $prob(p_2=y_1|vg_2, v_2) = 4/5$;
$prob(p_3=succ|f_1, v_1, p_2=y_1)=4/5$, $prob(p_3=succ|f_2, v_1, p_2=y_1)=3/5$,
$prob(p_3=succ|f_1, v_1,p_2=y_2)=1/2$, $prob(p_3=succ|f_2,v_1,p_2=y_2)=1/10$,
$prob(p_3=succ|f_1, v_2,p_2=y_1)=1/10$, $prob(p_3=succ|f_2,v_2, p_2=y_1)=2/5$,
$prob(p_3=succ|f_1, v_2,p_2=y_2)=2/5$, $prob(p_3=succ|f_2,v_2,p_2=y_2)=4/5$;

sensor: $prob(p_2=z_1|sd_1, s_1) = 4/5$, $prob(p_2=z_1|sd_2, s_1) = 1/5$,
$prob(p_2=z_1|sd_1, s_2) = 1/5$, $prob(p_2=z_1|sd_2, s_2) = 4/5$;
$prob(p_3=succ|c_1, s_1, p_2=z_1)=4/5$, $prob(p_3=succ|c_2, s_1, p_2=z_1)=3/5$,
$prob(p_3=succ|c_1, s_1,p_2=z_2)=1/2$, $prob(p_3=succ|c_2,s_1,p_2=z_2)=1/10$,
$prob(p_3=succ|c_1, s_2,p_2=z_1)=1/10$, $prob(p_3=succ|c_2,s_2, p_2=z_1)=2/5$,
$prob(p_3=succ|c_1, s_2,p_2=z_2)=2/5$, $prob(p_3=succ|c_2,s_2,p_2=z_2)=4/5$;

motion: $prob(p_3=succ|l_1, m_1)=4/5$, $prob(p_3=succ|l_2, m_1)=1/5$,
$prob(p_3=succ|l_1, m_2)=3/10$, $prob(p_3=succ|l_2, m_2)=4/5$;
$prob(p_7=succ|k_1, h_1)=4/5$, $prob(p_7=succ|k_2, h_1)=1/5$,
$prob(p_7=succ|k_1, h_2)=3/10$, $prob(p_7=succ|k_2, h_2)=4/5$;

The average costs of the translations under different situations can be calculated as

vision: $J(vg_1|v_1)=0.2^*$, $J(vg_2|v_1)=0.8$; $J(vg_1|v_2)=0.8$, $J(vg_2|v_2)=0.2^*$;
$J(f_1|v_1,y_1)=0.2^*$, $J(f_1|v_1,y_2)=0.5^*$; $J(f_1|v_2,y_1)=0.9$, $J(f_1|v_2,y_2)=0.6$;
$J(f_2|v_1,y_1)=0.4$, $J(f_2|v_1,y_2)=0.9$; $J(f_2|v_2,y_1)=0.6^*$,$J(f_2|v_2,y_2)=0.2^*$;

sensor: $J(sd_1|s_1)=0.2^*$, $J(sd_2|s_1)=0.8$; $J(sd_1|s_2)=0.8$, $J(sd_2|s_2)=0.2^*$;
$J(c_1|s_1,z_1)=0.2^*$, $J(c_1|s_1,z_2)=0.5^*$; $J(c_1|s_2,z_1)=0.9$, $J(c_1|s_2,z_2)=0.6$;
$J(c_2|s_1,z_1)=0.4$, $J(c_2|s_1,z_2)=0.9$; $J(c_2|s_2,z_1)=0.6^*$,$J(c_2|s_2,z_2)=0.2^*$;

motion: $J(l_1|m_1)=0.2^*$, $J(l_2|m_1)=0.8$; $J(l_1|m_2)=0.7$, $J(l_2|m_2)=0.2^*$;
$J(k_1|h_1)=0.2^*$, $J(k_2|h_1)=0.8$; $J(k_1|h_2)=0.7$, $J(k_2|h_2)=0.2^*$;

where J(w|s) is the average cost of the translation w under the situation s and the minimum average costs at the given situations are marked with *. These average costs indicate that when the situation v_1 is observed, the translation vg_1 (of t_2 in the vision coordinator) should be selected, ..., etc. Since the average costs are actually unknown, the optimal translations have to be realized through learning by the coordinators.

Since the probabilities of feedbacks are changing with the learning processes in the coordinators, the average costs for the translations in the dispatcher cannot be calculated. However, the limit distributions of the feedbacks, that is, the distributions of the feedbacks after the optimal translations in the coordinators have been learnt, can be found to be

vision: $prob(succ|v_1) = 0.8\times0.8 + 0.5\times0.2 = 0.74$,

$prob(succ|v_2) = 0.8\times0.8 + 0.4\times0.2 = 0.72$;

sensor: $prob(succ|s_1) = 0.8\times0.8 + 0.5\times0.2 = 0.74$,

$prob(succ|s_2) = 0.8\times0.8 + 0.4\times0.2 = 0.72$;

motion: $prob(succ|m_1) = 0.8$, $prob(succ|m_2) = 0.8$;

$prob(succ|h_1) = 0.8$, $prob(succ|h_2) = 0.8$;

dispatcher: t_2: $prob(c_1|m_1v_1)=prob(c_1|m_1v_2)=prob(c_1|m_1)=prob(succ|m_1) = 0.8$,

$prob(c_1|m_2v_1)=prob(c_1|m_2v_2)=prob(c_1|m_2)=prob(succ|m_2) = 0.8$;

t_5: $prob(c_1|v_1s_1)=prob(c_1|v_2s_1)=prob(c_1|s_1)=prob(succ|s_1) = 0.74$,

$prob(c_1|v_1s_2)=prob(c_1|v_2s_2)=prob(c_1|s_2)=prob(succ|s_2) = 0.72$;

t_6: $prob(c_1|h_1s_1)=prob(c_1|h_1s_2)=prob(c_1|h_1)=prob(succ|h_1) = 0.8$,

$prob(c_1|h_2s_1)=prob(c_1|h_2s_2)=prob(c_1|h_2)=prob(succ|h_2) = 0.8$;

assuming that in the translation string sets u_1, u_2, and u_3, every string in a set is selected with the equal probability, we can find that, for the dispatcher

t_2: $prob(c_1|u_1)=prob(c_1|u_2)=prob(c_1|u_3)= 0.8$;

t_5: $prob(c_1|u_1) = 0.74$, $prob(c_1|u_2) = 0.72$;

t_6: $prob(c_1|u_1)=prob(c_1|u_2)=prob(c_1|u_3)= 0.8$;

Therefore, assuming the learning processed in the coordinators have been completed, the average costs for the translations in the dispatcher now can be calculate as:

t_1: $J(v_1|p_1=n)=1\times0.8\times0.74+2\times(0.8\times0.26+0.2\times0.74)+5\times0.2\times0.26=1.564^*$,

$J(v_2|p_1=n)=1\times0.4\times0.72+2\times(0.4\times0.28+0.6\times0.72)+5\times0.6\times0.28=2.216$;

$J(v_1|p_1=s)=1\times0.4\times0.74+2\times(0.4\times0.26+0.6\times0.74)+5\times0.6\times0.26=2.172^*$,

$J(v_2|p_1=s)=1\times0.2\times0.72+2\times(0.2\times0.28+0.8\times0.72)+5\times0.8\times0.28=2.528$;

t_2: $J(u_1|n_1)=1\times0.8\times0.8+2\times(0.8\times0.2+0.2\times0.8)+5\times0.2\times0.2=1.48^*$,

$J(u_2|n_1)=1\times0.6\times0.8+2\times(0.6\times0.2+0.4\times0.8)+5\times0.4\times0.2=1.76$,

$J(u_3|n_1)=1\times0.4\times0.8+2\times(0.4\times0.2+0.6\times0.8)+5\times0.6\times0.2=2.04$;

$J(u_1|n_2)=1\times0.6\times0.8+2\times(0.6\times0.2+0.4\times0.8)+5\times0.4\times0.2=1.76^*$,

$J(u_2|n_2)=1\times0.4\times0.8+2\times(0.4\times0.2+0.6\times0.8)+5\times0.6\times0.2=2.04$,

$J(u_3|n_2)=1\times0.2\times0.8+2\times(0.2\times0.2+0.8\times0.8)+5\times0.8\times0.2=2.32$;

$J(u_1|n_3)=1\times0.3\times0.8+2\times(0.3\times0.2+0.7\times0.8)+5\times0.7\times0.2=2.18^*$,

$J(u_2|n_3)=1\times0.2\times0.8+2\times(0.2\times0.2+0.8\times0.8)+5\times0.8\times0.2=2.32$,

$J(u_3|n_3)=1\times0.1\times0.8+2\times(0.1\times0.2+0.9\times0.8)+5\times0.9\times0.2=2.46$;

t_5: $J(u_1|p_4=n)=1\times0.8\times0.74+2\times(0.8\times0.26+0.2\times0.74)+5\times0.2\times0.26=1.564^*$,

$J(u_2|p_4=n)=1\times0.4\times0.72+2\times(0.4\times0.28+0.6\times0.72)+5\times0.6\times0.28=2.216$;

$J(u_1|p_4=s)=1\times0.4\times0.74+2\times(0.4\times0.26+0.6\times0.74)+5\times0.6\times0.26=2.172^*$,

$J(u_2|p_4=s)=1\times0.2\times0.72+2\times(0.2\times0.28+0.8\times0.72)+5\times0.8\times0.28=2.528$;

t_6: $J(u_1|n_1)=1\times0.8\times0.8+2\times(0.8\times0.2+0.2\times0.8)+5\times0.2\times0.2=1.48^*$,

$J(u_2|n_1)=1\times0.6\times0.8+2\times(0.6\times0.2+0.4\times0.8)+5\times0.4\times0.2=1.76$,

$J(u_3|n_1)=1\times0.4\times0.8+2\times(0.4\times0.2+0.6\times0.8)+5\times0.6\times0.2=2.04$;

$J(u_1|n_2)=1\times0.6\times0.8+2\times(0.6\times0.2+0.4\times0.8)+5\times0.4\times0.2=1.76^*$,

$J(u_2|n_2)=1\times0.4\times0.8+2\times(0.4\times0.2+0.6\times0.8)+5\times0.6\times0.2=2.04$,

$J(u_3|n_2)=1\times0.2\times0.8+2\times(0.2\times0.2+0.8\times0.8)+5\times0.8\times0.2=2.32$;

$$J(u_1|n_3)=1\times0.3\times0.8+2\times(0.3\times0.2+0.7\times0.8)+5\times0.7\times0.2=2.18^*,$$

$$J(u_2|n_3)=1\times0.2\times0.8+2\times(0.2\times0.2+0.8\times0.8)+5\times0.8\times0.2=2.32,$$

$$J(u_3|n_3)=1\times0.1\times0.8+2\times(0.1\times0.2+0.9\times0.8)+5\times0.9\times0.2=2.46;$$

The minimum average costs in the limit case at the given situations are marked with *. The minimum average costs indicate that, when the learning time for the coordinators is longer enough, for any situations, the transitions t_1, t_2, t_5, and t_6 should selected the first translation or the translations in the first translation set. Again, since the average costs are actually unknown, the optimal translations have to be realized through learning by the dispatcher. It should also be pointed out that *the learning of the dispatcher depends the learning of the coordinators.*

B. Simulation Results: The zero initial cost and the uniform initial subjective probabilities used for all the translations. Also, the β and γ of the form

$$\beta(k) = \gamma(k) = \frac{1}{k}$$

are used in the performance estimation and the subjective probability update algorithms for all the translations. Note that the β and γ selected satisfy Dvoretsky's convergence conditions.

The following four task strings from the Organization Level are used as the inputs to the dispatcher in the simulation

$$s_1=e_5e_4e_2e_5e_4e_3e_5e_4e_1,$$

$$s_2=e_5e_4e_2e_4e_3e_4e_1,$$

$$s_3=e_5e_4e_2e_5e_4e_3e_4e_1,$$

$$s_4=e_5e_4e_2e_4e_3e_5e_4e_1.$$

These strings are defined based on the example by Valavanis (1986). To process these strings, the initial mark of the dispatcher is settled up to be two tokens in the place p_1, and no token in all other places. Each of the two initial tokens can be either a normal token or a special token with the equal probability 0.5. The scheduling procedure described in the section 5.1 is used for the task scheduling in the simulation.

Figure 6.6-6.12 in Appendix give the simulation results of executing each of the tasks s_1, s_2, s_3, and s_4 ten times. The total number of task executions, therefore, is 40. Comparing the learning curves of the coordinators in Figure 6.6-12 and the learning curves of the dispatcher in Figure 6.13-18, we can see clearly that the learning speeds in the coordinators are much fast than the learning speeds in the dispatcher. This can be explained by the fact that the learning in the dispatcher depends on the learning in the coordinators. It seems that the dispatcher can learn its optimal translations only after the coordinators have learnt their optimal translations. Figures 6.19-6.22 describe the changes of the total pure translation entropies of the dispatcher and the coordinators. The situation distributions appeared in the calculation of the pure translation entropies are replaced by the corresponding frequencies in which situations occurred so far, since the real values are unknown (see the section 5.2 for their calculations). These Figures indicate that the learning can reduce the pure translation entropies. Again, the entropy reducing speed of the dispatcher is much slow than that of the coordinators. Note that the behaviors of the vision and the sensor coordinators are very similar, since the same structure, cost functions, and probabilities are used for both of them.

7. Conclusions and Future Researches

The work reported above is just the first step toward the establishment of a complete analytic model for the Coordination Level of Intelligent Machines. However, it does reflect the basic features of the model to be completed. Upon to this stage, the two most difficult issues appeared in the research are

1. The gap between the theory and the reality of current robotic systems. Since the controls at the Coordination Level of the known robotic systems are deterministic, it is not clear how to apply the probabilistic task translation for the current available robotic systems.

2. The establishment of an overall control formulation.The effort to use Entropy as the overall control formulations had been made, however, the desired result is not achieved. The entropy is used passively in this report, that is, just as an index to measure the learning process, not actively used as a performance index to guide the task translation. Therefore some overall control formulation for the Coordination Level is still to be found.

The following research focuses are suggested to be on

1. Develop a stepwise refinement method for the design of bounded and live Petri net models for the dispatcher and coordinators.

Some results have already been obtained. We hope to used this stepwise method to expand the models constructed in the case study, therefore to make a more realistic simulations;

2. Investigate the scheduling problem with time factor [Shen 1988];

3. Use the concepts developed in [Lauer and Campbell 1975] and [Nehmer 1975] to build a general dispatcher, which is applicable to a class of coordination activities, and use the Petri net to model it;

4. See the possibility of introducing new learning algorithms.

5. Find some overall control formulations for the coordination level. Based on what we already obtained, the team theoretic formulation is the most promising one. However, the difficulties rely on the way of describing the effects of transition execution on each other and the chose of an overall cost function, as required in the team-like theories. The team formulation for integration, coordination of multi-sensor robot system in [Durrant-Whyte 1987] and the intelligent control method suggested in [Cruz and Stubberud 1987] are very good references in this regard.

The focus 5 will be the main research focus in the next semester.

BIBLIOGRAPHY

Aho, A.V. and Ullman, J.D. (1971) The Theory of Parsing, Translation and Compling, Englewood Cliffs, NJ: Prentice-Hall, 1972, Vol.1.

Albus, J.S. (1975) A New Approach to Manipulation Control: the Cerebellar Model Articulation Controller, *Transactions of ASME. J. of Dynamic Systems, Measurement and Control97*. pp220-227.

Al-Jaar, R. Y. and Desrochers, A.A. (1988), Petri Nets in Automation and Manufacturing, To appear in *Advances in Automation and Robotics*, Saridis, G.N. (ed.), volume 2. JAI Press Inc., Connecticut.

Al-Jaar, R. Y. and Desrochers, A.A. (1988), Modeling and Analysis of Transfer Lines and Production Networks Using Generalized Stochastic Petri Nets, *Proc. of the Conf. on University Programs in Computer-Aided Engineering, Design and Manufacturing*, pp. 12-21, Atlanta Georgia.

Al-Jaar, R. Y. and Desrochers, A.A. (1988), A Modular Approach for The performance Analysis of Automated Manufacturing Systems Using Generalized Stochastic Petri Nets, CIM Report #175, R.P.I., Troy, N.Y.

Antsaklis,P.J., Passino, K.M., and Wang, S.J. (1988), Autonomous Control Systems: Architecture and Fundamental Issues, *Proc. American Control Conf.*, Vol.1, pp602-607.

Baba, N. (1987) Learning Behaviors of the Hierarchical Structure Stochastic Automata Under the Nonstationary Multiteacher. Vol. SMC-17, No. 5, pp.868-872.

Barto, A.G. and Anandan, P. (1985) Pattern-Recognizing Stochastic Learning Automata, *SMC*, Vol.SMC-15, No.3, pp360-375.

Beck, C.L. and Krogh, B.H. (1986) Models for Simulation and Discrete Control of Manufacturing Systems, *87 IEEE International Robotics and Automation*, Vol.1, pp305-310.

Bejczy, A. (1985) Task Driven Control, *IEEE Workshop on Intelligent Control,1985*, pp.38. Rensselaer Polytechnic Institute, Troy, New York.

Boettcher, K.L. and Levis, A.H. (1982) Modeling the Interacting Decisionmaker with Bounded Rationality, *IEEE Trans. on Systems Science and Cybernetics*, Vol. SSC-12, No.3, pp334-343.

Bourbakis, N.G. (1987) TALOS - A Real-time, Ditributed Image Analysis/Synthesis System: Structural Design and Petri-net Modelling, *87 International Conf. on Industrial Electronics, Control and Instrumentation (IECON)*, pp556-560.

Brinch-Hansen, P. (1978), Distributed processes: A concurrent programming concept, *Comm. ACM*, Vol.21, No. 11, pp934-941.

Cai, Z.N., Farnhham, A., Ghalwash, A.Z., and Newcomb, R. (1987) Petri-Nets for Robot Lattices, *87 IEEE International Robotics and Automation*, Vol.2, pp999-1004.

Campbell, R.H. and Habermann, A.N. (1974), The specification of process synchronization by path expressions, in *Operating Systems* , LNCS, No.16, (E. Gelenbe and C. Kaiser, Eds.), pp89-102, Springer-Verlag, Berlin.

Chang, W.-H., Hyung, L.-K., Park, K.-H., and Kim, M.-N. (1987) State Characterization by Maximal Set of Concurrently Firable Actions in Petri Net Based Models, *Compters and Communications Technology Toward 2000*, pp472-476.

Chang, W.-H., Oh, H.-R., Hyung, L.-K., Park, K.-H., and Kim, M. (1988) Parallel Execution Schemes in a Petri Net , *88 International Conf. on Parallel Process (ICPP)*, pp286-290.

Corkill, D.D. (1979) Hierarchical Planning in a Distributed Environment, *Proc. 6th IJCAI*, pp168-175.

Corkill, D.D. and Lesser, V.R. (1983) The Use of fMeta-Level Control for Coordination in a Distributed Problem-Solving Network, *Proc. 8th IJCAI*, pp748-755.

Crockett, D., Desrochers, A., Dicesare, F. and Ward, T. (1987) Implementation of a Petri Net controller for a Machining Workstation, *Proc. of IEEE Robotics and Automation Conference*, pp1861-1867, Raleigh, North Carolina.

Cruz, J. B. Jr. and Stubberud, A. R. (1987) Performance Evaluation Using A Class of Petri Nets With Deterministic and Exponential Firing Times, *Computers and Communications Technology Toward 2000*, pp893-898.

Decker, K.S. (1987) Distributed Problem-Solving Technigues: A Survey, *SMC*, Vol.SMC-17, No.5, pp729-740.

Dijkstra, E. W. (1968a), The structure of "THE" multiprogramming system, *Comm. ACM*, 11, No. 5, pp341-346.

Dijkstra, E. W. (1968b), *Co-operating sequential processes*, in *Programming Languages*, (Ed. F. Genuys), pp.43-112, Academic Press, New York.

Durrant-Whyte, H.F. (1987) *Integration, Coordination and Control of Multi-Sensor Robot Systems*, Kluwer Academic Publishers.

Futo, I. and Gergely, T. (1983) Cooperative Problem Solving By Intelligent Actors, *Proc. IFAC Artificial Intelligence*, pp121-126.

Georgeff, M. (1984) A Thory of Action for MultiAgent Planning, *Proc. 9th IJCAI*, pp121-125.

Ghalwash, A.Z., Ligomenides, P.A., and Newcomb, R. (1987) Mutilayered Petri-Nets for Distributed Decision Making, *Proc. AFIPS National Computer Conf.*, Vol.2, pp257-263.

Ghalwash, A.Z. and Ligomenides,P.A. (1987) Colored Petri Net Analysis of Interactive and Concurrent Decision Organizations, *87 IEEE International Conf. on Systems, Man, and Cybernetics*, pp1091-1095.

This paper primarily deals with the second phase of decision making process (goal seeking) usinng Petri nets for representation and analysis of dm-organizations. In such organizations, each decision maker bas been assigned a specific task environment, and is guided by the search for the "best" nonotonic attainment of local goal, derived from the goals and constraints contained in the input command statements (during goal setting phase), and from the optiions of alternative actions available.

Ghalwash, A.Z. (1988) *Petri-Net Modeling of Decision Making Organizations*, Ph. D. Dissertation, EE Dept., Univ. Maryland, College Park, MD.

The decision making (dm) process for an organization is characterized, in general, by two phases, namely the "goal setting" and the "goal seeking" phases, repectively. A basic decision making model, characterized by two operational phases (goal setting/seeking) is introduced in the Dissertation. In the goal setting phase input command statements ripple down from dm activity at higher keveks, which modify the perception and specifications of the dm environment. The second phase is an image driven phase in which the decision maker receives images from the environment, or other organizational members, which denote a set of processes or subsystems. The dm process should control or act upon to achieve the goals prescribed by the goal setting phase. Those images relate recent, current and anticipated states and events, in the environments, for purposes of decision making and monitoring the effects of decision making. Output signals are actuation signaks responsible for physical action, through which the decision maker imparts control on his own environment, or subcommands that are directed to other organizational dm-members. Communications, in general, may include callouts, or data requests that transfer either state, event of status information. (These activities occur in the Coordination Level)

Graham, J.H. and Saridis, G.N. (1982) Linguistic Decision Structures for Hierarchical Systems, *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-12, No.3, pp323-333.

Hayward, V. and Hayti, S. (1988) KALI: An Environment for the Programming and Control of Cooperative Manipulators, *Proc. American Control Conf.*, Vol.1, pp473-478.

Heukeroth, V. and Nour Eldin, H.A. (1988) Interactive Systems: Knowledge Representation, Organization ans Automation, *Proceedings of IFAC/IMACS International Symposium on Distributed Intelligence Systems: Methods and Applications*, Varna, Bulgaria, pp71-77.

Hewitt, C. (1979), *Viewing Control Structures as Patterns of Passing Messages, Artificial Intelligence*, Vol.8, No.3, pp323-364.

Ho, Y.C. and Cassundras, C. (1983) A New Approach to the Analysis of Discrete Event Dynamical Systems, *IFAC Journal Automatica*, Vol.19, No.2, pp149-167.

Hoare, C.A.R. (1978) Communicating Sequential Processes, *Communications of the ACM*, Vol.21, No.8, pp666-677.

Hoare, C.A.R. (1985) *Communicating Sequential Processes*, Prentice-Hall International, Englewood Cliffs, NJ.

Hyung, L.-K. and Favrel, J. (1985) Hierarchical Reduction and Decomposition Method for Analysis and Decomposition of Petri Nets, *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-15, No.1, pp272-280.

Hyung, L.-K., Favrel, J. and Baptiste, P. (1987) Generalized Petri Net Reduction Method, *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-17, No.2, pp297-303.

Hyung, L.-K., Favrel, J., and Oh, G.-R. (1987) Hierarchical Decomposition of Petri Net Languages, *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-17, No.5, pp877-878.

Inan, K. and Varaiya, P. (1988) Finitely Recursive Process Models for Discrete Event Systems, *IEEE Trans. on Automatic Control*, Vol. AC-33, No.7, pp626-639.

Ionescu, D. and Trif, I. (1988) Hierarchical Expert System for Process Control, *Proc. American Control Conf.*, Vol.1, pp1-6.

Jaynes, E.T. (1955) Information Theory and Statistical Mechanics, *Physical Review*, Vol. 106, No.4, pp620-630.

Jaynes, E.T. (1968) Prior Probabilisties, *IEEE Trans. on Systems Science and Cybernetics*, Vol. SSC-4, No.3, pp227-241.

Jensen, K. (1981), Colored Petri Nets and the Invariant-method, *Theoretical Computer Science* , Vol.14, pp. 317-336, North-Holland.

Karsai,G., Biegl,C., Padalkar, Sztipanovits, Kawamura,K., Miyasaka,N., and Inui,M. (1988) Knowledge-Based Approach to Real-Time Supervisory Control, *Proc. American Control Conf.*, Vol.1, pp620-625.

Komoda, N., Kera, K., and Kubo, T. (1984), An Automated Decentralized Control System for Factory Automation , *IEEE Computer*, Vol.17, No.12, pp73-83.

Krauss, K.G. and Gulden, S.L. (1988) A Petri Net Method for the Formal Verification of Parallel Processes, *88 International Conf. on Parallel Process (ICPP)*, pp157-160.

Krogh, B.H. and Wilson,R., and Pathak, D. (1988) Automated Generation and Evaluation of Control Programs for Discrete Manufacturing Processes, *88 International Conf. on Computer Integrated Manufacturing,* Troy, N.Y., pp92-99.

Lauer, P.E. and Campbell, R.H. (1975) Formal Semantics of a Class of High-Level Primitives for Coordinating Concurrent Processes, *Acta Informatica,* Vol.5, Fasc.4, pp297-332.

High level programs for generating sustems of cooperating concurrent processes are classified according to syntactic criteria. Their semantic characterization by means of Petri Nets induces a corresponding syntactic classification on the latter. This permits the transferal of intuitively important semantical results of Petri Net theory to their corresponding programs. As a consequence it becomes possible to determine a certain kind of semantic correctness of a programm merely by its syntactic classification. Alternate solutions to a well known synchronization problem are treated in this way.

Lee, D.-H. and Kim, C.-S. (1987) Extended Timed Petri Net Model for Performance Analysis of Communication Protocols, *Compters and Communications Technology Toward 2000,* pp477-481.

Lee, I. and Goldwasser, S.M. (1985) A Distributed Testbed for Active Sensory Procesing, *Proceedings of 1985 Robotics and Automation,* pp925-930.

Lee,I., King,R., and X.Yun. (1988) Real-Time Kernel for Distributed Multi-Robot Systems, *Proc. American Control Conf.*, Vol.2, pp1083-1088.

Lesser, V.R. and Corkill, D.D (1981) Functionally-Accurate, Cooperatiive Distributed Systems, *SMC*, Vol.SMC-11, No.1, pp81-96.

Levis, A.H. (1988) Human Organizations as Distributed Intelligence Systems, *Proceedings of IFAC/IMACS International Symposium on Distributed Intelligence Systems: Methods and Applications*, Varna, Bulgaria, pp13-19.

Mandyam, A.L., Thathachar and Ramakrishnan, K.R. (1981) A Hierarchical System of Learning Automata., Vol. SMC-11, No. 3, pp.236-241.

Milne, G. and Milner, R. (1979), Concurrent Processes and Their Syntax, *J. Assoc. Comput. Mach.*, Vol.26, No.2, pp302-321.

Milner, R. (1980) *A Calculus of Communicating Systems*, Springer Verlag, New York.

Meystel, A. (1985). Intelligent Motion Control in Anthropomorphic Machines, Chapter in Applied Artificial Intelligence, S. Andriole ed., Prentrocellis Books. Princeton New Jersey.

Meystel, A. (1985). Intelligent Motion Control in Anthropomorphic Machines, Chapter in Applied Artificial Intelligence, S. Andriole ed., Prentrocellis Books. Princeton New Jersey.

Murata, T. (1980) Reduction and Expansion of Live and Safe Marked Graphs, *IEEE Trans. on Circuits and Systems*, Vol. CAS-27, No.1, pp.

Murata, T., N. Komoda, and K. Matsumoto (1986), A Petri-Net Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation, *IEEE Trans. on Industrial Electronics*, Vol. IE-33, No. 1, pp. 1-8.

Narahari, Y., and Viswanadham, N. (1987) Performance Evaluation Using A Class of Petri Nets With Deterministic and Exponential Firing Times, *Computers and Communications Technology Toward 2000*, pp482-486.

Nehmer, J. (1975) Dispatcher Primitives for the Construction of Operating System Kernels, *Acta Informatica*, Vol.5, Fasc.4, pp237-255.

Pang, G.K.H. (1988) A Blockboard Control Architecture for Real-Time Control, *Proc. American Control Conf.*, Vol.1, pp221-226.

Pao, Y.H. (1986) Some Views on Analytic and Artificial Intelligence Approaches, *Proc IEEE Workshop on Intelligent Contr.*, pp29, RPI., Troy, N.Y.

Passino, K.M. and Antsaklis, P.J. (1988a) Artifical Intelligence Planning Problems in a Petri Net Framework, *Proc. American Control Conf.*, Vol.1, pp626-631.

Passino, K.M. and Antsaklis, P.J. (1988b) Planning via Heuristic Search in a Petri Net Framework, *Proceedings of Third IEEE International Intellgent Control Symposium*, ppxxx, Arlington, VA.

Peterson, J.L (1973) *Modeling of Parallel Systems*, Ph.D, Thesis, EE Department, Stanford Univ., Stanford, CA.
This thesis dealed with computer operation systems (e.g., an operating system dealing with a disk system).

Peterson, J.L (1976) Computation Sequence Sets, *J. Computer and System Sciences*, Vol.13, No.1, pp1-24.

Peterson, J.L (1980) A Note on Colored Petri Nets, *Information Processing Letters*, Vol.11, No.1, pp40-43.
The use of colored tokens in a Petri net model can allow a much more concise model of a system. As long as the number of colors is finite, the model is equivalent to a (much large) Petri net without colors. However, allowing an infinite number of colors results in an extended model equivalent to a Turing machine, for which most general questions are undecidable.

Peterson, J.L (1981) *Petri Net Theory ans The Modeling of Systems*, Prentice-Hall International, Englewood Cliffs, NJ.

Petri, C.A. (1973), Concepts of Net Theory, in *Mathematical Foundations of Concepts of Computer Science*, LNCS, No.5, pp137-146, Springer-Verlag, Berlin.

Pnueli, A. (1979), *The Temporal Semantic of Concurrent Programs*, LNSC, No.70, Springer-Verlag, Berlin.

Reghizzi, S.C. (1977) Petri Nets and Szilard Languages, *Inform. Contr.*, Vol.33, pp177-192.

Saridis, G.N. (1977) *Self-organization Controls of Stochastic Systems.*, Marcel Dekker, New York.

Saridis, G.N and Stephanou, H.E. (1977) A Hierarchical Approach to the Control of A Prosthetic Arm , *SMC*, Vol.SMC-7, No.6, pp407-420.

Saridis, G.N. (1979) Toward Realization of Intelligent Control, *Proc. IEEE*, Vol.67.

Saridis, G.N. (1980) Intelligent Control for Advanced Automated Processes, in *Conf. on Automated Decision Making and Problem Solving*, NASA CP-2180, Langley.

Saridis, G.N. (1983) Intelligent Robotic Control, *IEEE Trans. Automatic Contr.*, Vol.AC-28, No.5, pp547-557.

Saridis, G.N. and Graham, J.H. (1984) Lingustic Decision Schemata for Intelligent Robots, *IFAC Journal Automatica*, Vol.20, No.1, pp121-126.

Saridis, G.N. (1985a) Foundations of Intelligent Controls, *Proc.IEEE Workshop on Intelligent Contr.*, pp23-27, RPI., Troy, N.Y.

Saridis, G.N. and Valavnis, K.P. (1985b) Information Theoretic Approach for Knowledge Engineering and Intelligent Machines, *Proc. ACC Conf.*, Boston, MA, pp.

Saridis, G.N. (1987) Knowledge Implementation: Structures of Intelligent Control Systems, *Proc. 2nd IEEE Workshop on Intelligent Contr.*, pp9-17.

Saridis, G.N (1988b) Analytic Formulation of the Principle of Increasing Precision with Decreasing Intelligence for Intelligent Machines,

Saridis, G.N. (1988c) Intelligent Machines: Distributed vs. Hierarchical Intelligence, *Proceedings of IFAC/IMACS International Symposium on Distributed Intelligence Systemns: Methods and Applications*, Varna, Bulgaria, pp34-39.

Smith, R.D. (1981a) *A Framework for Distributed Problem Solving*, Ph.D Thesis, UMI Press, Ann Arbor, MI.

Smith, R.D. (1981b) The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver, *IEEE Trans Comput.*, Vol.C-29, No.4, pp1104-1113.

Smith, R.D. and Davis, R. (1981) Frameworks for Cooperation in Distributed Problem Solving, *SMC*, Vol.SMC-11, No.1, pp61-70.

Smith, R.D. and Davis, R. (1983) Negotiation as A Metaphor for Distributed Problem Solving, *Artificial Intelligence*, Vol.20, No.1, pp63-109.

Shen, S. (1988) Cooperative Distributed Dynamic Load Balancing, *Acta Informatica*, Vol.25, Fasc.6, pp663-676.

Soog, J.-S., Satoh, S. and Ramamoorthy, C.V. (1987) The Abstraction of Petri Net, *Compters and Communications Technology Toward 2000*, pp467-471.

Steenstrup, M., Arbib, M.A., and Manes, E.G. (1983) Port Automata and the Algebra of Concurrent Processes, *J. Computer and System Sciences*, Vol. 27, No.1, pp29-50.

Stotts, P.D. (1988) The PFG Environment: Parallel Programming With Petri Net Semantics, *88 Hawaii International Conf. on System Sciences*, pp630-638.

Suzuki, I. and Murata, T. (1980) A Method for Hierarchically Representing Large Scale Petri Nets, *Proc. 1980 Int. Conf. on Circuits and Computer*, pp497-506.

Suzuki, I. and Murata, T. (1982) Stepwise Refinements of Transition and Places, Informatik-Fachberichet 52: *Application and Theory of Petri Nets*, Ed. Giraultt, C. and Reisig,W., pp136-141.

Suzuki, I. and Murata, T. (1983) A Method for Stepwise Refinements and Abstractions of Petri Nets, *J. Computer and System Sciences*, Vol. 27, No.1, pp51-76.

Trakhtenbrot, B.A., Rabinovich, A, and Hirshfeld, J. (1988) Nets of Processe, Technical Report, School of Mathematical Sciences, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel-Aviv University.

Valavnis, K.P. (1986) *A Mathematical Formulation for the Analytical Design of Intelligent Machines*, RAL Report #85, 294 pages.

Valette, R. (1979) Analysis of Petri Nets by Stepwise Refinements, *J. Computer and System Sciences*, Vol. 18, No.1, pp35-46.

Valk, R. and Vidal-Naquet, G. (1981) Petri Nets and Regular Languages, *J. Computer and System Sciences*, Vol. 23, No.x, pp299-325.

Vamos, T. (1987) Metalanguages-Conceptusl Model: Bridge Between Machine and Human Intelligence, *Proc. 1st Internationsl Symposium on AI and Expert Systeds*, pp237-287.

Wang, F.-Y. and Saridis, G. N. (1988) Structural Formulation of Plan Generation for Intelligent Machines,

Wang, F.-Y. and Saridis, G.N. (1988) A Formal Model for Coordination of Intelligent Machines using Petri Nets, *Proceedings of Third IEEE International Intellgent Control Symposium*, ppxxx, Arlington, VA.

Yang, J.Y.D., Huhns, M., and Stephens, L.M. (1985) An Architecture for Control and Communications in Distributed Artificial Intelligence Systems, *SMC*, Vol.SMC-15, No.3, pp316-326.

Yau, S.S. and Chou, C.-R. (1988) Control Flow Analysis of Distributed Computing System Software Using Structured Petri Net Model, *Workshop on the Future Trends of Distributed Computing in the 1990s*, , pp174-183.

Zhou, M.-C. and DiCesare, F. (1988) Adaptive Design of Petri Net Controllers for Automatic Error Recovery, *Proceedings of Third IEEE International Intellgent Control Symposium*, ppxxx, Arlington, VA.

Zhou, M. C., F. DiCesare, and A. A. Desrochers (1988), A Formal Methodology for Synthesis of Petri Net Models and Controllers for Manufacturing Systems, Working Paper, ECSE Dept., Rensselaer Polytechnic Institute, August 1988.

Appendix

Simulation Results: Figures 6.6-22

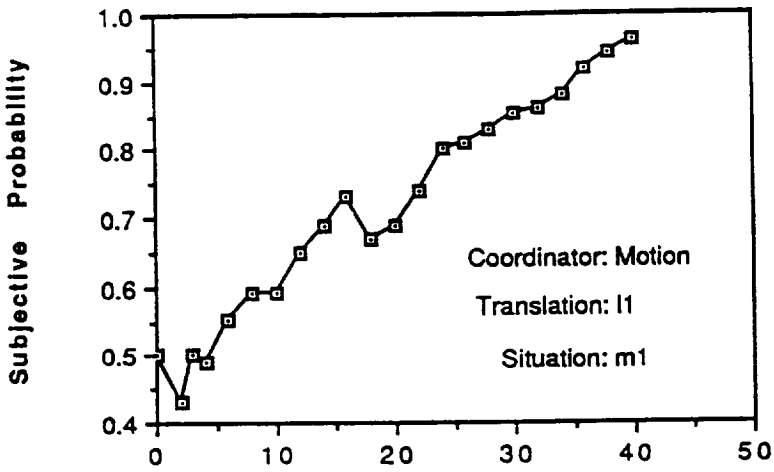

Fig. 6.6 Number of Executions

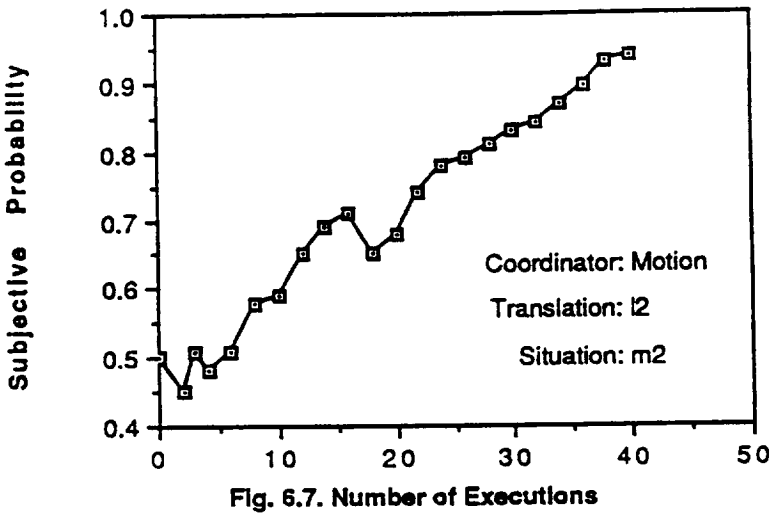

Fig. 6.7. Number of Executions

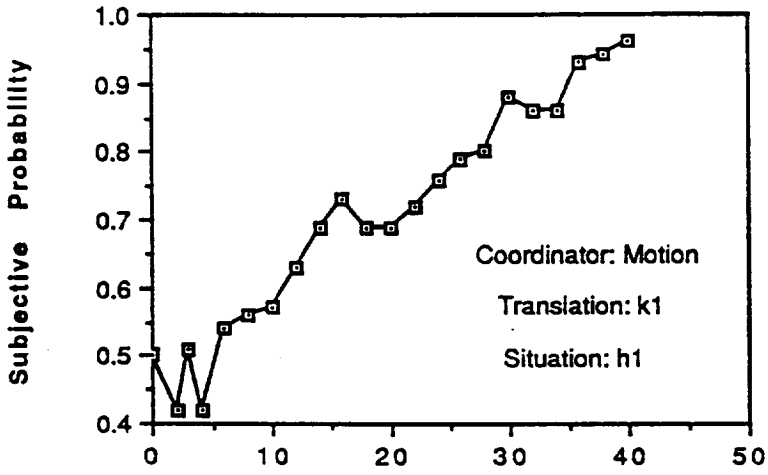

Fig 6.8 Number of Executions

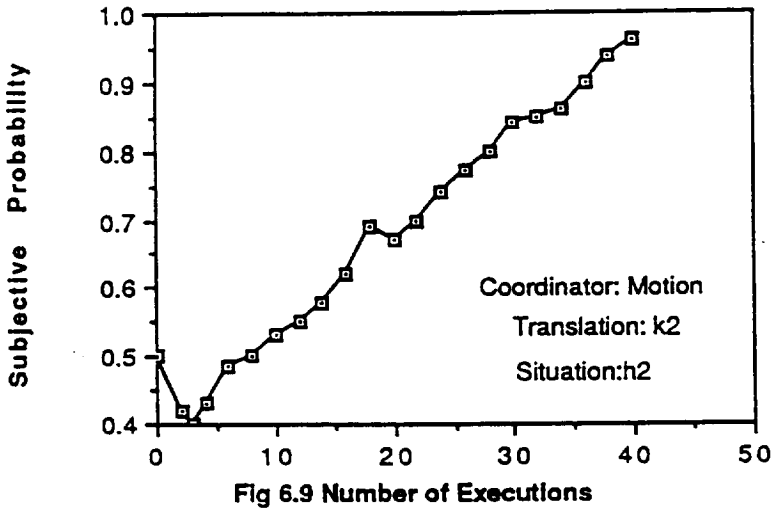

Fig 6.9 Number of Executions

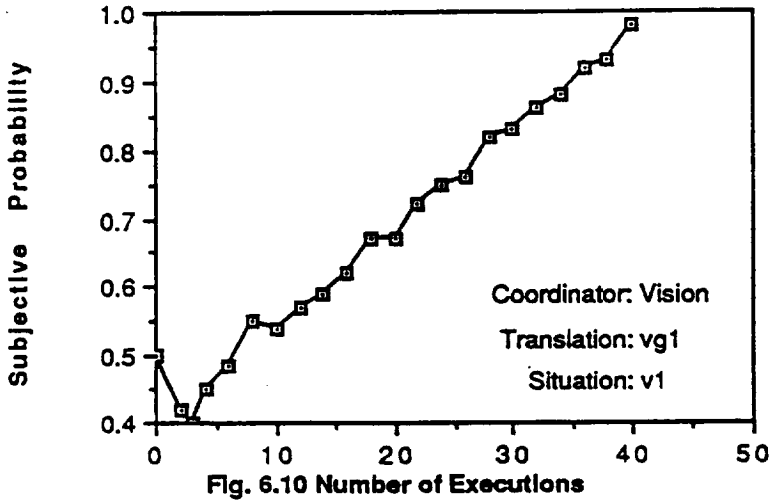

Fig. 6.10 Number of Executions

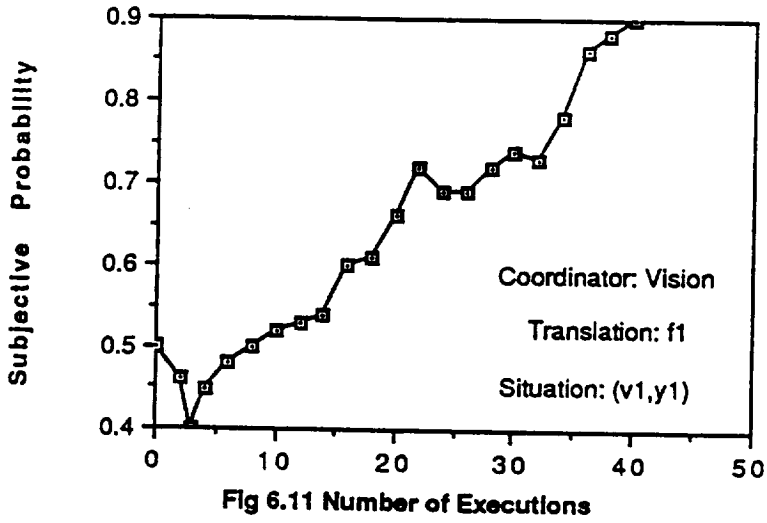

Fig 6.11 Number of Executions

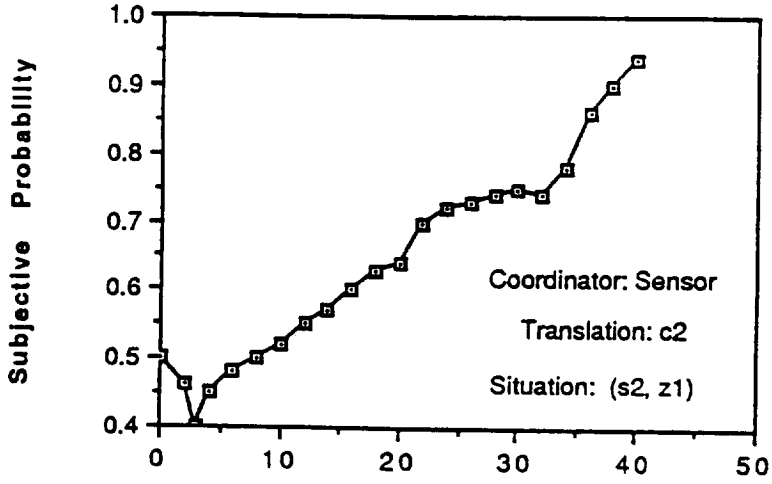

Fig. 6.12 Number of Executions

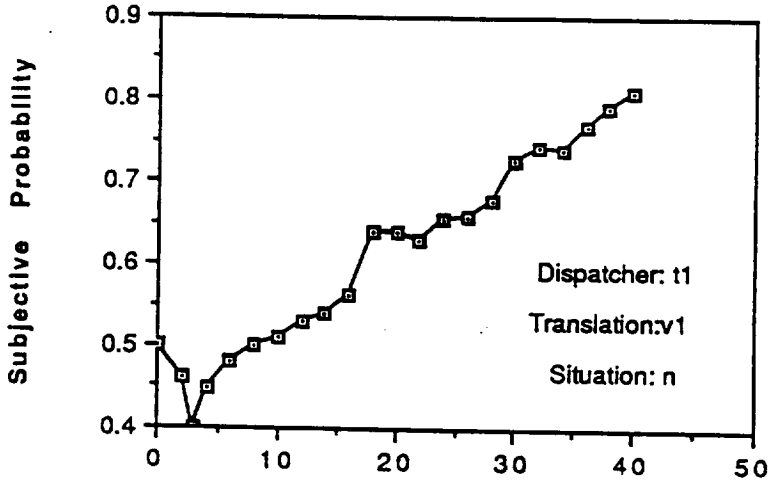

Fig 6.13 Number of Executions

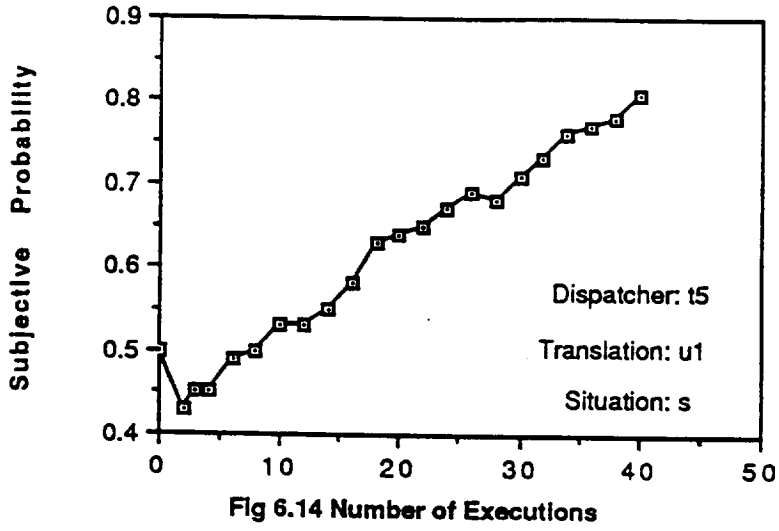

Fig 6.14 Number of Executions

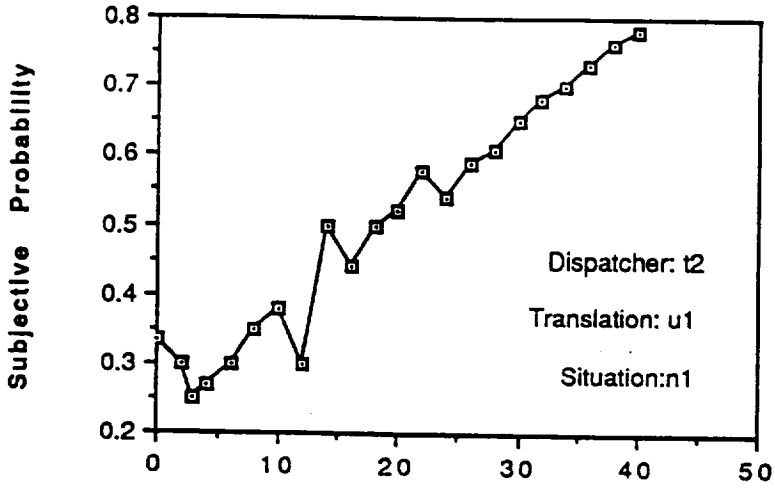

Fig 6.15 Number of Executions

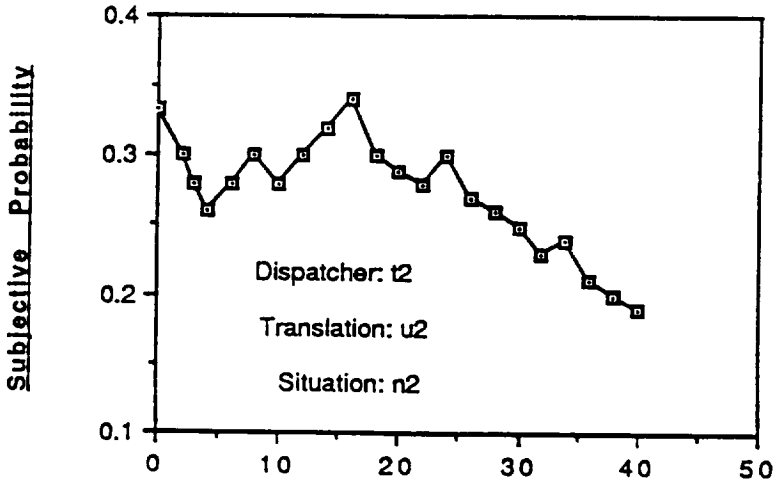

Fig 6.16 Number of Executions

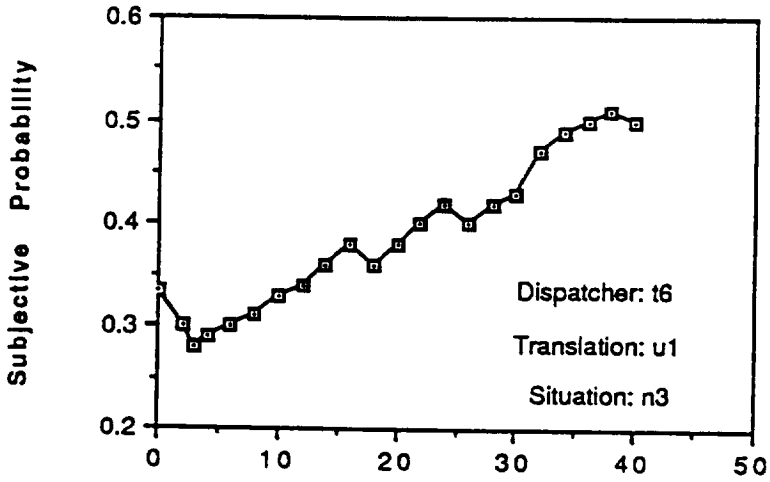

Fig. 6.17 Number of Executions

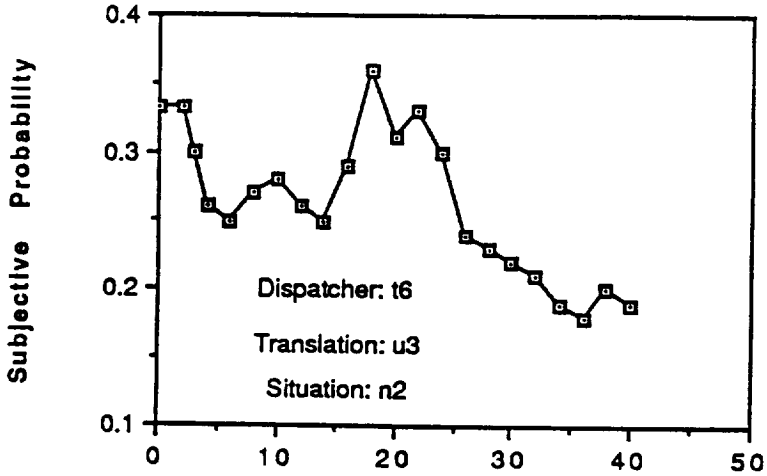

Fig 6.18 Number of Executions

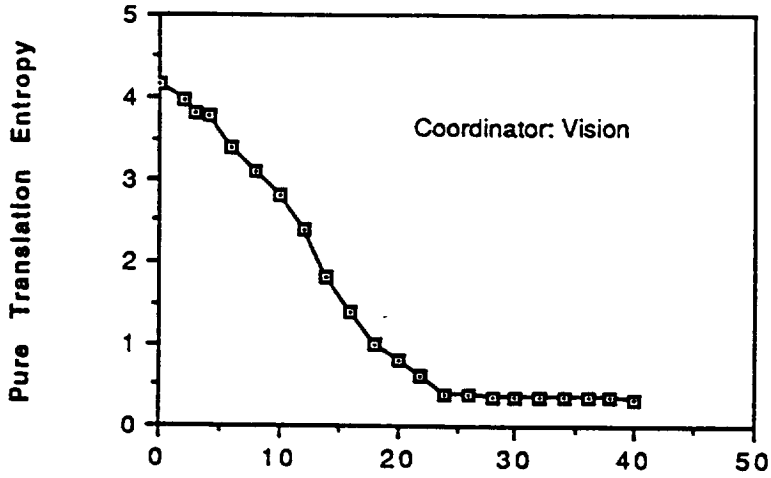

Fig. 6.19 Number of Execution

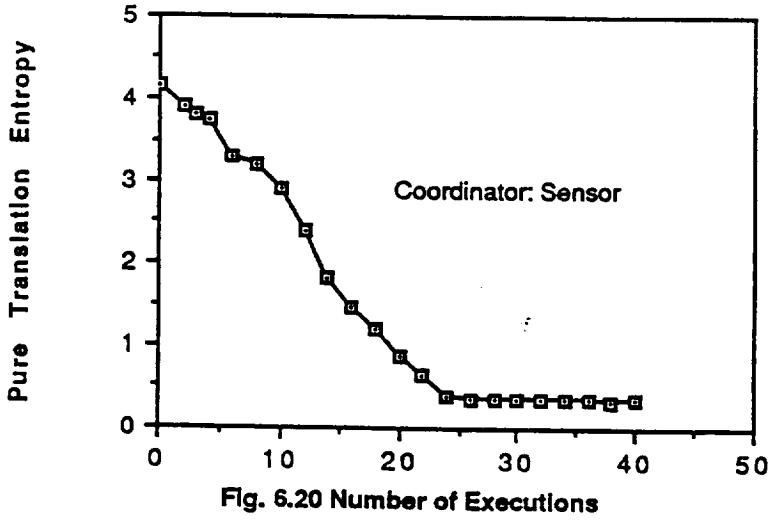

Fig. 6.20 Number of Executions

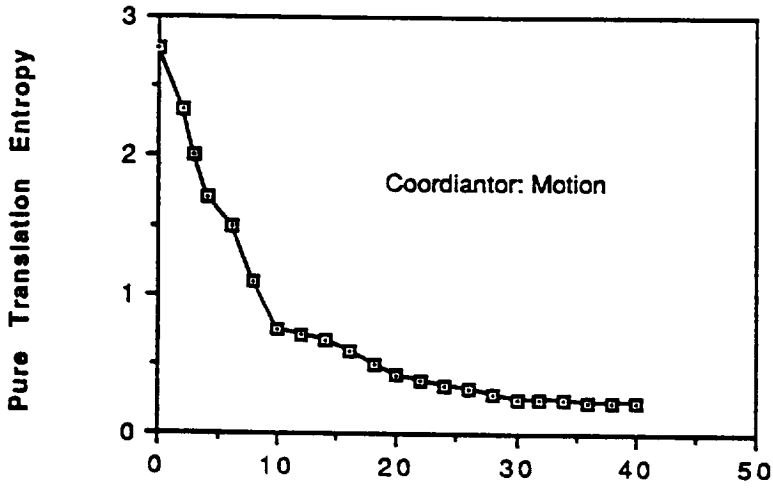

Fig. 6.21 Number of Executions

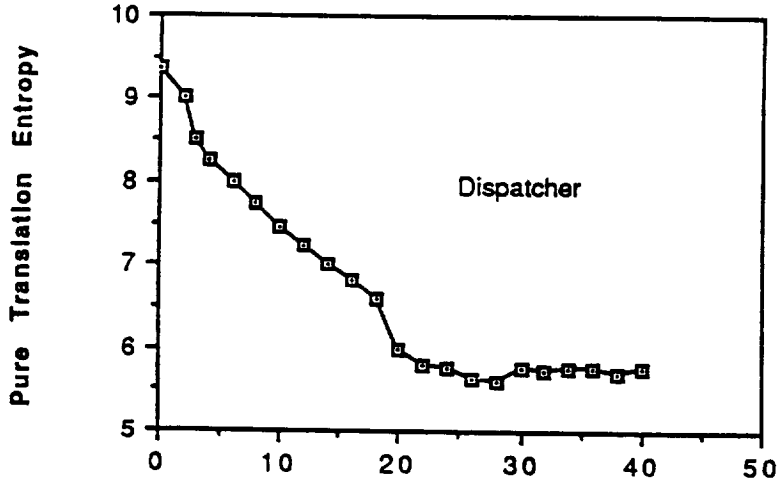

Fig 6.22 Number of Executions